View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Edinburgh Research Archive

Unsupervised Learning of Multiple Objects in Images

Michalis K. Titsias

Doctor of Philosophy

Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh

2005

Abstract

Developing computer vision algorithms able to learn from unsegmented images containing multiple objects is important since this is how humans constantly learn from visual experiences. In this thesis we consider images containing views of multiple objects and our task is to learn about each of the objects present in the images. This task can be approached as a factorial learning problem, where each image is explained by instantiating a model for each of the objects present with the correct instantiation parameters. A major problem with learning a factorial model is that as the number of objects increases, there is a combinatorial explosion of the number of configurations that need to be considered. We develop a greedy algorithm to extract object models sequentially from the data by making use of a robust statistical method, thus avoiding the combinatorial explosion.

When we have video data, we greatly speed up the greedy algorithm by carrying out approximate tracking of the multiple objects in the scene. This method is applied to raw image sequence data and extracts the objects one at a time. First, the (possibly moving) background is learned, and moving objects are found at later stages. The algorithm recursively updates an appearance model so that occlusion is taken into account, and matches this model to the frames through the sequence. We apply this method to learn multiple objects in image sequences as well as articulated parts of the human body. Additionally, we learn a distribution over parts undergoing full affine transformations that expresses the relative movements of the parts.

The idea of fitting a model to data sequentially using robust statistics is quite general and it can be applied to other models. We describe a method for training mixture models by learning one component at a time and thus building the mixture model in a sequential manner. We do this by incorporating an outlier component into the mixture model which allows us to fit just one data cluster by "ignoring" the rest of the clusters. Once a model is fitted we remove from consideration all the data explained by this model and then repeat the operation. This algorithm can be used to provide a sensible initialization of the mixture components when we train a mixture model.

Acknowledgements

I am grateful to my supervisor Chris Williams for his invaluable guidance. I would also like to thank David Barber for his advice in the early stage of the PhD. The last three years I have been benefitted greatly from discussions with other members of the ANC in Edinburgh and also from meetings of the machine learning group. Especially, I thank Felix Agakov and Michael Schouten for innumerable discussions. Finally, I would like to thank my family for their support especially the period of writing this thesis.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Michalis K. Titsias*)

Dedicated to the memory of my father Konstantinos Titsias.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents a framework for unsupervised learning of multiple objects in images. This chapter serves as an introduction to the topic of this thesis. In section 1.1 we give a definition of the general problem that computer vision attempts to solve and discuss why learning from examples can be useful for doing this. We also discuss supervised and unsupervised learning and how these methods have been used for object recognition in computer vision. In section 1.2 we give a chapter by chapter outline of the thesis.

1.1 Computer vision and learning from examples

Computer vision faces the problem of giving interpretation to images similarly to the ability of the human visual system and brain to understand from visual experiences. A computer vision system need not restrict itself to mimic human or animal vision and can include, for example, other sensors such as those used in medical imaging technology. Additionally, computer vision can provide a computational solution to a problem that may not be similar to how human vision solves a similar problem. However, computers' performance is generally quite limited compared to human vision when both attempt to solve the same problem. Therefore, better understanding of human vision can potentially have a great impact on how computer algorithms are designed.

An illustrative way to think about how an imaging system will be used in practice

is to consider that it operates over time so that at each time it receives an input image or an image sequence and then answers questions about the phenomena depicted in the image or sequence. For example, such questions can be what is the category of each object viewed in the image (e.g. is this object a car or not), where an object is located, which is the area in the image that an object occupies, what an object is doing in the image (e.g. is this man running or walking) etc. The answers to these questions can be expressed as a set of discrete values or labels, so the system assigns labels to the image that "explain" what is viewed. For example, if we are interested in labelling the image according to the category of the objects it contains, the labels might be 'human-face' and 'car' indicating the image contains human faces and cars. We can have additional labels such as labels expressing how many different faces and cars there are, separate labels for each image pixel indicating the object the pixel is part of and others.

An ideal computer vision system can be constructed as follows. Enumerate all possible images and for each image let the system memorize all the answers that we wish to provide. However, this solution is computationally infeasible. There are far too many images, e.g. for 20×20 gray scale images with 256 gray levels there are 256^{400} different combinations. The space of all possible labellings can be also very large and thus intractable to naively enumerate.

Although the number of images is large, they are not randomly generated but highly structured and constrained. This structure can be thought as a kind of repeatability property that obtains several forms. To mention some examples, if we observe a patch in an image with certain colour or texture is very likely this patch to be repeated (but not with exactly the same appearance) in a neighbouring area of the image. Also if we have a set of images showing faces, even though all faces are different there are repeated characteristics that exist in any face image. Similar images will obtain similar labellings and thus there is an underlying smooth relationship between input images and desired labellings. This allows for a principled solution based on learning from a set of training images. For example, say we have a set of images of faces and we wish to learn how to discriminate between faces and any other object. A solution based on learning from examples will be to use a set of images showing human faces and attempt to memorize the 'essence' of a face or in other words what is roughly

repeated in any face image and use this information to detect faces in novel images. Learning from examples does require memorization, however, we might not need to explicitly memorize the training images but extract some important information from these examples that can be transferred to novel images.

Learning from examples has long history in statistics and pattern recognition; see e.g. Duda and Hart (1973). One commonly applied set of techniques for image analysis are **supervised** classification algorithms where the objective is to learn how to predict the class label of an input vector through a training process using a set of labelled examples. Object recognition is a classification problem where an object view should be classified into one of a set of possible categories. However, when there are multiple objects instantiated in an image, multiple class labels need to be provided. Additionally, due to the relative locations of camera and object, the size of the object within the image as well as the pose and the orientation can vary and this variation can generate a very large number of different images of the same object. Thus, the exact process for training a classifier from a set of images needs careful consideration. A widely used training framework is to collect a set of training images with each image showing a single object (against some background) and label each image with the class label of the object shown (Ullman, 1996; Forsyth and Ponce, 2003). Furthermore, the images are normalized so that the object has a certain size and location (usually centred in the image) so that the classifier does not have to deal with the huge variation due to the object's instantiation parameters. Collecting the training images requires significant human involvement. Typically, a human should either create the images in a suitable imaging environment where one object appears in the images (with additional constraint of being centred and with a certain size) or segment object views from more naturally captured images containing multiple objects. The way the trained classifier is applied to novel data also needs consideration, since we wish to classify naturally taken images with multiple objects and account also for occlusion and clutter rather than one-object per image examples as those used in the training. This process, also called recognition or detection, is desired to be efficient and fast.

An different class of learning methods is **unsupervised** algorithms which make use only of the input data without requiring any labelling information. In unsuper-

vised learning we aim to identify regularities in data. One fairly simple unsupervised learning model is clustering, which assumes that each input data comes from a finite number of types of object, and thus data is produced by choosing one of these objects and then generating the data conditional on this choice. For learning objects in images general purpose clustering algorithms presented in the literature are not immediately useful. A reason for this is that in order to discover objects by clustering we should be able to deal with the variation due to the instantiation parameters. For example, a useful clustering algorithm should be able to group images of the same object into one cluster independently from the variation due to translation, rotation and scaling. Learning about objects taking this regularity into account has been called transformation-invariant clustering by Frey and Jojic (1999, 2003). However, this work is still limited to finding a single specific object (not an object category) from the training images. Furthermore, realistic images contain multiple objects and thus clustering is not very useful since it can only assign a single label to the image, while the image should be explained by multiple labels. A more general model explains an image by multiple causes that correspond to different objects. This type of unsupervised learning is referred as factorial learning, see, for example, Barlow (1989), Hinton and Zemel (1994), Saund (1995). The approach of Frey and Jojic (1999) can be extended so that to deal with multiple objects (Jojic and Frey, 2001). The main focus of this thesis is to further investigate unsupervised factorial learning by providing a new probabilistic model and algorithm for learning multiple objects as well as object parts from images.

Humans seem to learn how to recognize objects in quite different learning scenarios than the one commonly used in computer vision for training a object recognition system. Particularly, we constantly learn new object categories from visual experiences containing multiple objects under clutter and occlusion, yet this ability seems to largely come from unsupervised learning. Particularly, our ability to group parts of an image into objects is probably based on cues such as motion, spatial coherence, texture similarity, the experience of similar objects seen in the past and others (Wertheimer, 1923; Kanizsa, 1979), see also Forsyth et al. (1997). For this reason, developing unsupervised learning algorithms and especially algorithms that learn from images with multiple objects under complex backgrounds and occlusion is important, and might

have a great impact on computer vision performance in the future.

1.2 Outline of the thesis

This thesis addresses the problem of learning multiple objects from a set of images using unsupervised learning. The input images we are dealing with show some specific objects, such as cars and humans, against some background. The objects and the background move relative to each other so that in different images an object can be transformed according to translation, rotation, scaling etc. Note that such data are frequently captured in image sequences.

Chapter 2 introduces a probabilistic generative model that explains each image as a synthesis of a background model and L models corresponding to L foreground objects. The object models are 2D appearance images and masks where the latter specify the shape of the objects. The model generates an image by first selecting transformations for the individual objects and the background and then synthesizing the image so that objects combine by occlusion. Jojic and Frey (2001) have developed a similar generative model and used a variational EM algorithm for training. The chapter also gives a brief introduction to object recognition since we view learning multiple objects as part of the general problem of learning an object recognition system using a set of example images.

Exact learning using the EM algorithm of the model for multiple objects is intractable. In **Chapter 3** we develop a sequential or greedy algorithm that finds one object at each time using a robust statistical method. The robust statistical method means that when we learn an object, certain parts of an image where other not-yet-explained objects exist are considered as outliers. Once we have learned an object, we have explained the area in any training image that is occupied by this object and thus by removing the respective image pixels from consideration we can search for a new object at the next run of the algorithm. In this chapter we show results of extracting objects in real images considering only a translational motion of the objects.

Chapter 4 builds upon the framework for learning multiple objects using the greedy algorithm and focuses on learning from video data as opposed to unordered images.

Particularly, this chapter describes the following methods:

- For video data we greatly speed up the greedy algorithm by tracking the objects before knowing their full structure. Note that tracking allows to compute the object's transformations quickly. We have developed a novel tracking algorithm that works in conjunction with the greedy algorithm.

- We apply this tracking algorithm to learn parts of articulated objects, such as the human body, from video data. Once the appearances of the parts of an object have been learned, we compute a joint probability distribution over the transformations of the parts that models the joint instantiation of these parts so that valid object's shapes can be generated. This distribution is derived by expressing the transformations through using transformed landmarks on the parts and computing a mixture of Gaussians distribution over these landmarks, where each Gaussian is a probabilistic Principal Components Analysis (PCA) model.

The methods in this chapter are demonstrated by experiments in real video sequences and considering full affine transformations for the objects.

The idea of using robust statistics and a greedy algorithm to learn a model in a sequential way is quite general and can be applied to others models. In **Chapter 5** we show how we can train a mixture model using an outlier component so that at each stage a new mixture component is fitted to the data. We show that this algorithm can be useful for providing a good initial parameter estimate when we train a mixture model.

Chapter 6 summarizes the main contributions of the thesis and describes directions for future research.

Chapter 2

Learning multiple objects from images

We start this chapter by giving an introduction to the object recognition problem. Then, we describe a probabilistic model for learning appearance-based models of multiple objects from a set of images using unsupervised learning. The generative model produces an image as a synthesis of L foreground objects, modelled by 2D appearance images and masks, and a background which is modelled by an appearance image. In this chapter we do not describe any algorithm for learning the parameters of this model as this is the topic of chapter 3. Much of the work of this chapter as well as the next chapter has been presented in Williams and Titsias (2003, 2004).

Section 2.1 gives a brief introduction to the problem of visual object recognition. Section 2.2 describes a probabilistic generative model for learning multiple objects and and the chapter concludes with a discussion in section 2.3.

2.1 Object recognition

Visual object recognition deals with the problem of detecting objects in images and assigning them into object classes. An object class can be a specific object such as the face of a certain individual, or more generally, an object category such as the class of all human faces and cars. Clearly, object recognition requires building a classifier like those developed in statistical pattern recognition, see e.g. Duda and Hart (1973). However, object recognition and image analysis have a number of idiosyncrasies that

make the classification process a very difficult and challenging problem.

To illustrate the important challenges that arise when developing an object recognition system, assume that images are constrained to have a neutral background and contain the view of a single object. We further assume that the object can sometimes be a member of some class "C". The system should respond with a positive answer when an image contains an object of class "C" and a negative answer otherwise. Clearly, the efficiency of this system depends on its ability to generalize across the space of variation according to which instances of the object class "C" can vary from image to image. There are two main sources of variation of object appearance in the images. The first source is related to the camera viewpoint and any other imaging conditions such as lighting conditions. Since an image is a projection of an three-dimensional world (with three-dimensional objects) into a two-dimensional space, the viewpoint of the camera can significantly affect the appearance of an object within the image. Due to this source of variation the object can be located in the image according to some unknown 3D transformation parameters. For example, consider pictures of a specific chair taken from different viewpoints; the images might vary significantly despite the fact the chair has invariant characteristics. The second source of variation describes changes related to the internal characteristics of the object class, so that even with a fixed camera viewpoint and imaging conditions the object appearance within the image can still vary. Such variation is called intra-class variation and includes changes of the object's shape, or any other kind of deformation, e.g. consider a cat running, a human face that changes expressions etc. Notice that when the object class is a general category rather than a specific object, large variability naturally arises due to the different specific objects falling within the same class.

Realistic images can have background clutter and contain multiple foreground objects that we wish to recognize. In this case the appearance of an object can vary because of possible occlusions or shadows caused by other objects in the scene. For example, imagine a camera tracking a man walking in a street where other objects such as cars and trees are interposed between the camera and the man, so that certain parts of the man completely disappear in an observed image. This type of variation caused by occlusion is different from the types of variation described earlier and can be thought

as a kind of structured noise. Also note that multiple objects can significantly complicate the recognition or detection of the objects in an novel image. Particularly, to detect multiple objects we should be able to carry out a segmentation of the image into different objects so that the segmented features that belong to a single object will be further classified to one of a number of possible classes.

Below we briefly review two general frameworks for building object recognition systems following the division into global and local methods adopted by Forsyth and Ponce (2003). Particularly, in section 2.1.1 we discuss global methods where the classifier attempts to model the variation in a global sense. In section 2.1.2 we discuss local methods that use separate models to deal with the variation of local parts of the object and have an additional model describing the co-occurrence or joint instantiation of the parts.

2.1.1 Global recognition methods

The simplest method for recognizing an instance of an object class in an image is template matching or matched spatial filters (see Brunelli and Poggio, 1995 for a relatively recent review). This method assumes that the object exists in some window of certain size within the image and matches all possible windows with a stored template. The template model is a 2D image showing the average intensity values of the object. If we don't know the orientation and scale of the object we need to consider many templates for each possible set of these parameters. Clearly, this approach can be only used for specific objects. since for general object classes, such as the classes of all faces, we cannot represent significant object intra-class variation with a template. More flexible templates are based on eigenspaces (Sirovich and Kirby, 1987; Turk and Pentland, 1991; Murase and Nayar, 1995). In this method a collection of objects within a class are represented as a linear combination of some basis templates estimated, for instance, as the first few principal components from a set of training images. The basis templates are then used as a set of global building blocks that represent the variation of the object class.

The techniques described so far can be thought as model-based methods since there is an assumed appearance model describing the object class and thus recognition is

finding a matching or correspondence of this model to a novel image. If this matching is good, then the object is present in the image, otherwise it is not. Also, this type of methods can be considered as generative classifiers since roughly speaking there is an underlying class-conditional distribution being modelled according to which an instance of the object class is generated given the set of parameters that describe the object model.

An alternative to the model-based approach is a general purpose discriminative classifier such as a neural network or a Support Vector Machine (SVM); see e.g. Bishop (1995); Vapnik (1998). For example, a neural network can be trained to output the posterior probability that an input image patch (or a set of features extracted from the original patch) depict the frontal view of a human face. Training can be carried out using labelled data of fixed-size image patches that consist of faces with (approximately) fixed orientation and scale as well as a set of negative examples coming from natural images that do not show a face. Once the neural network is trained, recognition is performed using the same principle as in the template matching; we scan the novel image by taking all windows of the same size as the image patch and compute the output of the neural network for this window (Poggio and Beymer, 1996). Rowley et al. (1998) applied such a method using neural networks for face recognition and similarly Oren et al. (1997) have used SVMs to recognize pedestrians.

2.1.2 Local recognition methods

Local based recognition methods make use of the idea that small parts of an object vary much less across instances of the object class than the whole object. Thus, the variation of the whole object can be modelled in a two-fold way: (i) model separately the variation of small parts and (ii) model the joint instantiation (e.g. spatial relationships) of these parts. For example, for face recognition, the parts may be templates for the eyes, noise and mouth. Two faces might have very similar parts, but they might look different since the different spatial layout of these parts. Using parts-based representation for objects dates back to Fischler and Elschlager (1973) pictorial structures model. An influential work in this idea is the Recognition by Components method of Biederman (1987) where some volumetric primitives are used as universal 3D parts.

Instead of having appearance based parts we might have any kind of local features that described a local area of the object. Such features include local image patches, corner or texture patches (Weber et al., 2000; Ioffe and Forsyth, 2001), wavelet functions (Schneiderman and Kanade, 2004) and Gaussian derivative filters (Felzenszwalb and Huttenlocher, 2000). Ullman et al. (2002) have proposed a method that selects class-specific 2D object fragments as parts which are selected from the training images using mutual information.

The above description of local recognition methods is a model based approach. The object model consists of the set of parts or local features and a model that describes the relationships between the parts (the joint model for the parts) according to which valid objects can be produced. Note that some authors ignore the relationships of the parts and considered them as being independent (see e.g. Ullman et al., 2002; Csurka et al., 2004). A simple way to match the object into an image is first to detect independently features from the image and then find correspondences between these features and the object parts. This is the approach taken by Weber et al. (2000); Fergus et al. (2003). However, this not the optimal strategy since the parts are less distinctive than the whole object and detection of them can be difficult. A more principled approach is described in Felzenszwalb and Huttenlocher (2000, 2005) which uses the joint model of the parts to assist the detection of the features, so that detection of the features and recognition of the object are carried out simultaneously. In their method, the prior as well as the posterior distribution of the parts locations has a tractable tree structure and an efficient dynamic programming algorithm is applied.

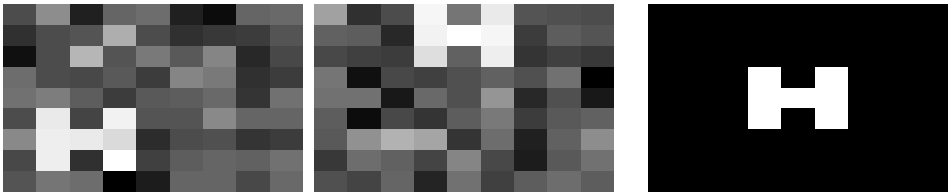
Parts-based recognition has also been applied in a discriminative framework. For example, Heisele et al. (2002) use separate SVMs to model the variation of object parts and uses an additional global SVM trained to work like a joint model for the parts. Particularly, for each object part they train a separate SVM to discriminate between instances of that part and any other image patch coming from natural images. Then, the global SVM takes as input the output of these SVMs and the locations of the detected parts and is trained to recognize the whole object.

An important question is how model-based or generative recognition systems compare with discriminative classifiers. Discriminative classifiers bypass a great amount

of object class variation and only concentrate on the decision boundary between the object class and the rest classes. However, when a novel image contains an object of some unknown class, the classifier will always choose one of the classes without having any obvious diagnostic mechanism indicating that the object is an "outlier". To overcome this, we should attempt to model the decision boundary of an object and all the rest natural images. However, we might need a enormous amount of negative examples (plus the positive examples) to efficiently discover such a decision boundary. The good point about these methods is that there are now several standard techniques such as SVMs, that someone can use as "black boxes". Generative methods attempt to model the internal variation of an object class and can be trained by using examples of only the object class of interest. This is an implicit way to approximate the decision boundary of the object class with the rest possible natural images, without requiring any reference to negative examples. However, views of an object class might have a huge variation and designing generative methods capable of expressing this variation can be very challenging. Finally, an advantage of the generative or model-based classifiers is that they can be applied even in case the class labels are not given. On the other hand discriminative classifiers will need the class labels at least for some of the examples.

2.2 Unsupervised learning of multiple objects

Training an object recognition system using a set of examples is typically carried out using labelled data. The notion of labelled data can take several forms. For example, some learning algorithms might require the objects in each image to be segmented or each image to contain a single object in a normalized location. Typically we will need a human either to do the segmentation or collect the images in the desired format. Also classifiers (and especially discriminative classifiers) need the class label of each training example. Parts-based methods might require additional labelling information, e.g. the methods of Felzenszwalb and Huttenlocher (2000); Heisele et al. (2002) need the label and the location of each part in all training examples. Developing less supervised training algorithms for object recognition can be difficult. For example, to automate



(a) (b)

Figure 2.1: Learning "H" shapes from noisy images. (a) displays two of the training images and (b) shows the true underlying appearance $\mathbf{f}$ of the object.

segmentation in still images with multiple objects captured in realistic imaging environments is a very hard problem.

In this section and in the most of the remaining part of the thesis we focus on methods that can learn object models from images using unsupervised learning. Such a method has been recently introduced by Frey and Jojic (1999, 2003) and it can learn a simple appearance template of an object from a set of images using a probabilistic generative model. In section 2.2.1 we review the Frey and Jojic work for learning a single object and we also discuss the advantages of probabilistic methods for unsupervised learning over other k-means type of algorithms. In sections 2.2.2 and 2.2.3 we discuss learning multiple objects.

2.2.1 Learning one object

Consider an image $\mathbf{x}$ of size $P_x \times P_y$ containing $P \stackrel{def}{=} P_x P_y$ pixels, arranged as a length P vector. Suppose now we have a set of noisy images $X = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ each containing the view of a foreground object in front of a highly varied (or clutter) background so that the location of the object within an image can change in different images. Figure 2.1a shows two examples of a artificial data set of 9×9 "H" shapes against a clutter background. We wish to learn the appearance of "H" using the available images. Next we view learning the object as the inversion of a generative process according to which each image was generated.

Let assume that the appearance of the object is an image $\mathbf{f}$ arranged as a length P vector. Figure 2.1b shows the underlying appearance corresponding to the "H" shapes.

An image $\mathbf{x}$ can be thought as being generated from $\mathbf{f}$ according to the following process:

- select a transformation j from a set of J possible transformations and apply j to $\mathbf{f}$ so that $T_j\mathbf{f}$ is the transformed appearance and T_j is the transformation matrix
- generate $\mathbf{x}$ by adding some zero mean independent noise to each pixel of $T_j\mathbf{f}$, where the noise variance can vary in different pixels

We assume that the set of transformations for the "H" shapes that can generate any possible image is a window of 7×7 translations in units of one pixel with wraparound. In this case the transformation matrix T_j is a permutation matrix that shifts the image $\mathbf{f}$ in the 2D plane. For example, the left image in Figure 2.1a is generated by shifting $\mathbf{f}$ shown in Figure 2.1b two pixels left and down and then adding noise. In general, the set of transformations can correspond to translations, rotations and scalings and T_j is generally a very sparse matrix. Latter in the thesis we will make use of transformation matrices that have only one non zero element (with value 1) in each row and correspond to an affine transformation of an image. The permutation matrices we assume here is a special case of such matrices. Appendix A.1 clarifies the form of these transformation matrices.

The appearance $\mathbf{f}$ is not known beforehand and our task is to learn this appearance using the set of training images. We can imagine this procedure as applying the inverse transformation according to which each training image was originally generated and averaging to obtain $\mathbf{f}$. However, this is not an easy task since the transformations are unknown and in order to specify them we need an estimate of $\mathbf{f}$[1]. A simple algorithm to solve this "chicken-and-egg" problem is to initialize $\mathbf{f}$ and iterate between the following two steps:

- given the current value of $\mathbf{f}$ find for each image $\mathbf{x}^n$ the transformation j^n that gives the best alignment (with the smallest euclidean distance) of $T_{j^n}\mathbf{f}$ with $\mathbf{x}^n$
- given the estimated transformations apply the inverse transformations to $\mathbf{x}^n$ and

[1] A good estimate of $\mathbf{f}$ would allow us to align $\mathbf{f}$ with each image and determine the transformations.

update $\mathbf{f}$

$$\mathbf{f} = \frac{1}{N}\sum_{n=1}^{N} T_{j^n}^{-1}\mathbf{x}^n, \tag{2.1}$$

where T_j^{-1} is the inverse transformation matrix, and since we have chosen T_j to be a permutation matrix, it is equal to the transpose T_j^T. We can easily show that the above algorithm minimizes the sum of square errors

$$E = \sum_{n=1}^{N} min_j ||\mathbf{x}^n - T_j\mathbf{f}||^2, \tag{2.2}$$

where $||\mathbf{y}-\mathbf{z}||^2$ denotes the squared euclidean distance between $\mathbf{y}$ and $\mathbf{z}$, with respect to $\mathbf{f}$ and the transformations. From now on we call this algorithm k-means algorithm since it operates similarly to the k-means used for clustering.

Clearly, the way an image $\mathbf{x}$ is generated from $\mathbf{f}$ is stochastic. A formal way to describe this stochastic process is to explicitly express a model for the probability density of $\mathbf{x}$. Next, we describe such a probabilistic model and we demonstrate that training this model using the EM algorithm is much more effective than the k-means algorithm.

Suppose first that the transformation j is selected according to some prior probability P_j so that $\sum_{j=1}^{J} P_j = 1$. Typically we don't have preference to any transformation and thus we can choose an uniform prior. Then, an image is generated by a Gaussian with spherical covariance matrix $\sigma_f^2 I$ and mean $T_j\mathbf{f}$. The choice of the spherical covariance matrix reflects our assumption that the noise is added independently to each pixel and additionally it has the same variance. Compiling together these assumptions the underlying density of $\mathbf{x}$ is the following mixture of Gaussians

$$p(\mathbf{x}) = \sum_{j=1}^{J} P_j N(\mathbf{x}; (T_j\mathbf{f}), \sigma_f^2 I), \tag{2.3}$$

where $N(\mathbf{x};\boldsymbol{\mu},\Sigma)$ denotes a Gaussian with mean $\boldsymbol{\mu}$ and covariance Σ. This model has been introduced by Frey and Jojic (1999, 2003)[2] and is referred as transformation-invariant clustering. To find the object appearance we can maximize the log likelihood

[2]To be precise, this is a simplification of the Frey and Jojic (2003) model where we assume that there is no noise added to $\mathbf{f}$ before the transformation is applied and also the post transformation noise has spherical covariance matrix.

$L = \sum_{n=1}^{N} \log p(\mathbf{x}^n)$ with respect to the parameters $(\mathbf{f}, \sigma_f^2)$. The EM algorithm (Dempster et al., 1977) can be used to find a local maximum of the log likelihood. Particularly given some initialization of $(\mathbf{f}, \sigma_f^2)$, EM iterates between the following steps:

- Expectation-step: Given the current values of $(\mathbf{f}, \sigma_f^2)$ find the posterior distribution over transformations $P(j|\mathbf{x}^n)$ for any image $\mathbf{x}^n$:

$$P(j|\mathbf{x}^n) = \frac{P_j N(\mathbf{x}^n; (T_j\mathbf{f}), \sigma_f^2 I)}{\sum_{i=1}^{J} P_i N(\mathbf{x}^n; (T_i\mathbf{f}), \sigma_f^2 I)}. \tag{2.4}$$

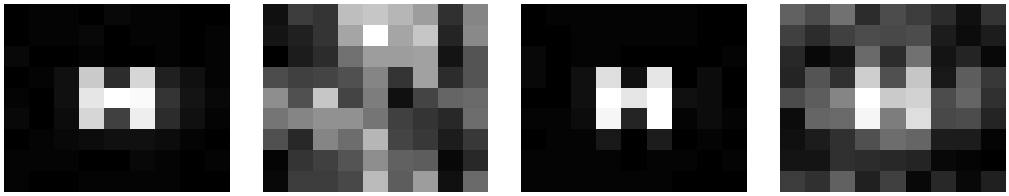
- Maximization-step: Given the weights $P(j|\mathbf{x}^n)$ update the parameters as follows:

$$\mathbf{f} = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{J} P(j|\mathbf{x}^n) T_j^{-1} \mathbf{x}^n, \tag{2.5}$$

$$\sigma_f^2 = \frac{1}{NP} \sum_{n=1}^{N} \sum_{j=1}^{J} P(j|\mathbf{x}^n) ||\mathbf{x}^n - T_j\mathbf{f}||^2. \tag{2.6}$$

How to derive these updates is shown in appendix A.2. Note that the EM algorithm iteratively optimizes over parameters by first expressing a distribution over all transformations and then updates the parameters using weighted averages based on the posteriors $P(j|\mathbf{x}^n)$. As the number of iterations increases typically each $P(j|\mathbf{x}^n)$ will become sharply picked around the most probable transformation and will obtain close to zero value for the rest of transformations. In practice, at the convergence point only one transformation, say j^n, will get all the probability mass (i.e. $P(j^n|\mathbf{x}^n) \simeq 1$) so that the update (2.5) will conform to the update of (2.1) of the k-means algorithm. However, at the early iterations the EM updates can be very different than those of k-means algorithm.

To compare the EM algorithm with the k-means we created 20 datasets of "H" shapes, each having 40 training examples. In each data set we choose an initialization for the appearance $\mathbf{f}$ so that each pixel value takes a random value within the range of maximum and minimum pixel value of all images. Then, we use the same initialization to run both algorithms. σ_f^2 for the EM is initialized to a large value. Figures 2.2a and 2.2b display the appearances found by the two algorithms averaged over the 20 datasets. We also repeat the above experiment by initializing $\mathbf{f}$ closer to the true value



(a) (b) (c) (d)

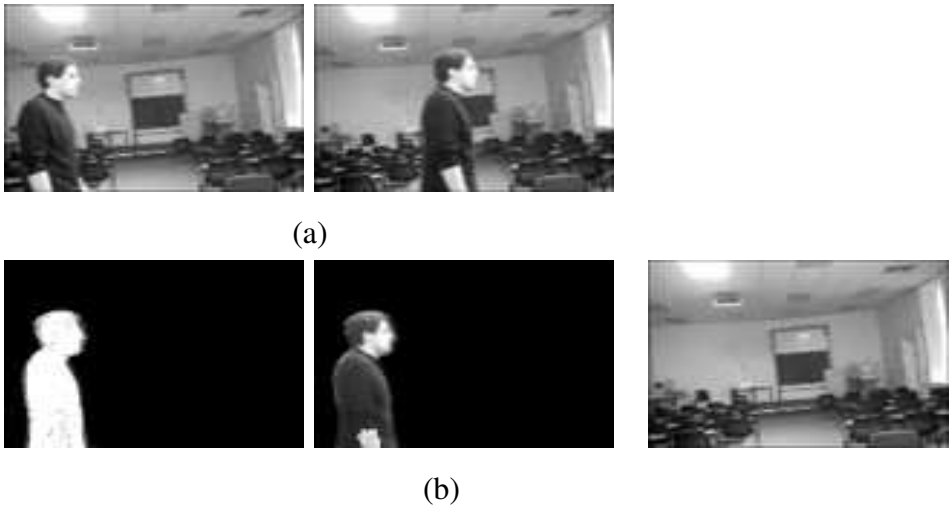
Figure 2.2: (a) shows the estimated appearance (averaged over 20 trials) for the EM algorithm using random initializations (within the range of the image pixel values) and (b) shows the corresponding appearance for the k-means. (d) and (c) show the estimated appearances for the EM and the k-means respectively that are found by initializing the appearances to be closer to the true value.

of the appearance and the results are displayed in Figure 2.2d for the EM and in Figure 2.2c for the k-means respectively. Clearly, EM can be bootstrapped by a random initialization of the object appearance $\mathbf{f}$, while k-means cannot find any meaningful answer for such initialization. Even with an improved intialization scheme k-means performs poorly.

Additionally to the above benefits over k-means type of algorithms for learning objects, probabilistic models allows us to introduce prior knowledge in a principled manner. For example, in the above model we can incorporate priors over transformations and over the parameters. Furthermore, we can always enhance a probabilistic model by plugging in new hidden variables and parameters so that to deal with more complex data. For example, what we will do in the next section is to modify the above probabilistic model in order to deal with images with one foreground object against a static or moving background rather than a clutter.

2.2.2 Learning one foreground object and the background

Consider the images of Figure 2.3a which show one foreground object against a static background. The model described in the previous section cannot efficiently explain these images since it assumes only one object present in the images while now the objects are clearly two: the background and the movable foreground object. Thus, we



(a)

(b)

Figure 2.3: One object moving against a static background. (a) panel shows two images of the training set, while panel (b) shows the object and the background found by the EM algorithm. The left plot in (b) shows the mask $\boldsymbol{\pi}$, the middle plot shows the element-wise product $\boldsymbol{\pi} * \mathbf{f}$ and the third plot displays the background $\mathbf{b}$.

should introduce an appearance model for the background as well. We will generally assume that the background can be one of three cases: (i) a static background that is fixed for all training images, (ii) a moving background which occurs for example when a moving camera captures a sequence of frames and (iii) random backgrounds where each image can have a completely different background.

The two key issues that we must deal with are the notion of a pixel being modelled as foreground or background, and the problem of transformations of the object and the background. We consider first the foreground/background issue and we assume that the background is static; cases (ii) and (iii) are discussed later in this section.

As the foreground object will occupy only a subset of the $P_x \times P_y$ pixels and the rest will be background, we will need to specify which pixels belong to the background and which to the foreground; this is achieved by a vector of binary latent variables $\mathbf{s}$, one for each pixel. Each binary variable in $\mathbf{s}$ is drawn independently from the corresponding entry in a vector of probabilities $\boldsymbol{\pi}$. For pixel p, if $\pi_p \simeq 0$, then the pixel will be ascribed to the background with high probability, and if $\pi_p \simeq 1$, it will be ascribed to the foreground with high probability. We sometimes refer to $\boldsymbol{\pi}$ as a mask since it

captures the shape of the object being learned.

x_p is modelled by a mixture distribution:

$$x_p \sim \begin{cases} p_f(x_p; f_p) = N(x_p; f_p, \sigma_f^2) & \text{if } s_p = 1, \\ p_b(x_p; b_p) = N(x_p; b_p, \sigma_b^2) & \text{if } s_p = 0, \end{cases} \tag{2.7}$$

where σ_f^2 and σ_b^2 are respectively the foreground and background variances. Thus, ignoring transformations, we obtain

$$p(\mathbf{x}) = \prod_{p=1}^{P} [\pi_p p_f(x_p; f_p) + (1 - \pi_p) p_b(x_p; b_p)]. \tag{2.8}$$

Notice that the fact that each pixel follows a mixture distribution ensures that the foreground and background appearances strictly combine by occlusion and thus no transparency between them is allowed.

The second issue that we must deal with is that of transformations of the foreground object. Similarly to the previous section we introduce a transformation variable j_f represented by a transformation matrix, so that matrix T_{j_f} corresponds to transformation j_f and $T_{j_f}\mathbf{f}$ is the transformed foreground model. The semantics of foreground and background mean that the mask $\boldsymbol{\pi}$ must also be transformed, so that we obtain

$$p(\mathbf{x}|j_f) = \prod_{p=1}^{P} [(T_{j_f}\boldsymbol{\pi})_p p_f(x_p; (T_{j_f}\mathbf{f})_p) + (\mathbf{1} - T_{j_f}\boldsymbol{\pi})_p p_b(x_p; b_p)], \tag{2.9}$$

where $\mathbf{1}$ denotes the P length vector that contains ones. Notice that the foreground $\mathbf{f}$ and mask $\boldsymbol{\pi}$ are transformed by T_{j_f}, but the static background $\mathbf{b}$ is not. In order for equation (2.9) to make sense, each element of $T_{j_f}\boldsymbol{\pi}$ must be a valid probability (lying in $[0, 1]$). This is certainly true for the case when T_{j_f} is a permutation matrix (and can be true more generally). To complete the model we place a prior probability P_{j_f} on each transformation j_f; this is taken to be uniform over all possibilities so that $p(\mathbf{x}) = \sum_{j_f=1}^{J_f} P_{j_f} p(\mathbf{x}|j_f)$. Clearly, the $p(\mathbf{x})$ is a mixture model similarly to that described by equation (2.3), with the difference that the conditional densities $p(\mathbf{x}|j_f)$ are not Gaussians but more complex densities that incorporate the background appearance as well. Note that if we constrain the mask $\boldsymbol{\pi}$ to be the vectors of ones, then the model will reduce to that described by (2.3).

The application of the EM algorithm becomes now a bit more complicated than the case of section 2.2.1. Particularly, in the E-step for each image $\mathbf{x}^n$ we need to estimate the posterior probability for each transformation $P(j_f|\mathbf{x}^n)$ according to

$$P(j_f|\mathbf{x}^n) = \frac{P_{j_f} p(\mathbf{x}^n|j_f)}{\sum_{i=1}^{J_f} P_i p(\mathbf{x}^n|i)}, \tag{2.10}$$

as well as a soft segmentation of the image by specifying a P length vector $\overline{\mathbf{s}}_{j_f}^n$ of probability values so that the pth element stores the value

$$(\overline{\mathbf{s}}_{j_f}^n)_p = \frac{(T_{j_f}\boldsymbol{\pi})_p p_f(x_p^n; (T_{j_f}\mathbf{f})_p)}{(T_{j_f}\boldsymbol{\pi})_p p_f(x_p^n; (T_{j_f}\mathbf{f})_p) + (\mathbf{1} - T_{j_f}\boldsymbol{\pi})_p p_b(x_p^n; b_p)}. \tag{2.11}$$

$(\overline{\mathbf{s}}_{j_f}^n)_p$ expresses the probability that the pth pixel of the image $\mathbf{x}^n$ is part of the foreground object given the transformation j_f. In the M-step we update the parameters $(\mathbf{f}, \boldsymbol{\pi}, \sigma_f^2, \mathbf{b}, \sigma_b^2)$. The update equations are given in the appendix A.2 but for example the update for $\mathbf{f}$ is

$$\mathbf{f} = \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n) T_{j_f}^T[\overline{\mathbf{s}}_{j_f}^n * \mathbf{x}^n] ./ \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n) T_{j_f}^T[\overline{\mathbf{s}}_{j_f}^n], \tag{2.12}$$

where $\mathbf{y} * \mathbf{z}$ and $\mathbf{y}./\mathbf{z}$ stands for the element-wise product and element-wise division between two vectors, respectively. This update is quite intuitive. Consider the case when $P(j_f|\mathbf{x}) = 1$ for $j_f = j^*$ and 0 otherwise. For pixels which are ascribed to the foreground (i.e. $(\overline{\mathbf{s}}_{j^*}^n)_p \simeq 1$), the values in $\mathbf{x}^n$ are transformed by $T_{j^*}^T$ (which is $T_{j^*}^{-1}$ for permutation matrices). This removes the effect of the transformation and thus allows the foreground pixels found in each training image to be averaged to produce $\mathbf{f}$. Figure 2.3b displays the foreground and background appearance found after applying the EM algorithm using a training set of 31 78×104 images.

So far the background $\mathbf{b}$ was considered to be static. However in many cases, as for example when a video camera follows an object, the background can change from frame to frame. Next we generalize our method to deal with moving backgrounds.

To model a moving background we assume an underlying static background which is typically much larger than the input images. We sometimes refer to this large static background as panorama. When we generate an image a part of this panorama scene is selected and used as the current background of the image, similarly to Rowe and

Blake (1995). More specifically, we assume that the background $\mathbf{b}$ corresponds to an $M_x \times M_y$ image, where in general $M_x \geq P_x$ and $M_y \geq P_y$. $\mathbf{b}$ is represented as a M-dimensional vector with $M = M_x M_y$. We introduce a transformation variable j_b that explains how from the panorama $\mathbf{b}$ the background of a data image is selected. In our implementation we consider as possible backgrounds all the $P_x \times P_y$ image blocks (aligned to the axes of the background image) taken from any possible location within the panorama $\mathbf{b}$[3]. Clearly, j_b takes on $J_b = (M_x - P_x + 1)(M_y - P_y + 1)$ total values and a certain value j_b is represented by a $M \times P$ transformation matrix T_{j_b}, so that $T_{j_b}\mathbf{b}$ selects the appropriate image $P_x \times P_y$ block from $\mathbf{b}$.

The conditional density of an image given the transformation variables now becomes

$$p(\mathbf{x}|j_f, j_b) = \prod_{p=1}^{P} [(T_{j_f}\boldsymbol{\pi})_p p_f(x_p; (T_{j_f}\mathbf{f})_p) + (\mathbf{1} - T_{j_f}\boldsymbol{\pi})_p p_b(x_p; (T_{j_b}\mathbf{b})_p)], \qquad (2.13)$$

and the likelihood of an image $\mathbf{x}$ is $p(\mathbf{x}) = \sum_{j_f=1}^{J_f} \sum_{j_b=1}^{J_b} P_{j_f} P_{j_b} p(\mathbf{x}|j_f, j_b)$. Note also that when the background is static is expressed as a special case of the above model; by choosing the background $\mathbf{b}$ to have the same size as the data images there is only one possible value for j_b, so the background is static.

For random backgrounds we do not try to model the backgrounds explicitly, but simply use a large-variance Gaussian at each pixel, which can account for the large background variability. $\mathbf{b}$ is the mean of this Gaussian.

We can again use the EM algorithm to handle the hidden variables which is the transformations j_f and j_b and the binary variables $\mathbf{s}$ and optimize over the parameters. However, an exact EM algorithm requires a search over $J_f J_b$ possibilities which can be very demanding even for small images. In chapter 3 we describe a greedy training algorithm that deals separately with each transformation by learning first the background and then the foreground object. This algorithm is presented for the more general case of L foreground objects in chapter 3 and also in the appendix B.

[3] Of course the above model does not account for rotations or scaling of the background and it can only approximately model such kind of situations.

2.2.3 Learning multiple objects

Assume that each image contains L foreground objects. Similarly to the single object case each foreground object ℓ, with $\ell = 1, \ldots, L$ is modelled by a separate appearance $\mathbf{f}_\ell$ and mask $\boldsymbol{\pi}_\ell$. The background can be thought as the $L+1$th object having a mask $\boldsymbol{\pi}_b = \mathbf{1}$, since the background is present everywhere. For each foreground object ℓ we assume a transformation variable j_ℓ representing all possible translations. Below we assume a moving background where the transformation variable j_b is defined in section 2.2.2, however all derivations also apply for static or random backgrounds by simply ignoring the variable j_b. We should point out that Jojic and Frey (2001) proposed a similar probabilistic generative model for learning multiple objects. Differences with the model described below will be discussed in section 3.5.

It will be instructive to introduce the model for the case there are only two foreground objects. Assuming $L = 2$, an image $\mathbf{x}$ is generated by instantiating the transformation variables j_1, j_2 and j_b and then drawing $\mathbf{x}$ according to

$$\begin{aligned} p(\mathbf{x}|j_b, j_1, j_2) &= \prod_{p=1}^{P} \Big\{ (T_{j_1}\boldsymbol{\pi}_1)_p p_{f_1}(x_p; (T_{j_1}\mathbf{f}_1)_p) + (\mathbf{1} - T_{j_1}\boldsymbol{\pi}_1)_p \times \\ &\quad [(T_{j_2}\boldsymbol{\pi}_2)_p p_{f_2}(x_p; (T_{j_2}\mathbf{f}_2)_p) + (\mathbf{1} - T_{j_2}\boldsymbol{\pi}_2)_p p_b(x_p; (T_{j_b}\mathbf{b})_p)] \Big\}, \end{aligned} \tag{2.14}$$

where the p_{f_1}, p_{f_2} and p_b pixel densities are Gaussians given as in equation (2.7). Note that each image pixel follows a three component mixture distribution, so that with probability $(T_{j_1}\boldsymbol{\pi}_1)_p$ the pixel can belong to the first object, with probability $(\mathbf{1} - T_{j_1}\boldsymbol{\pi}_1)_p (T_{j_2}\boldsymbol{\pi}_2)_p$ to the second object and with the rest of probability to the background. The fact that the probabilities corresponding to the second object's pixels are always multiplied by $(\mathbf{1} - T_{j_1}\boldsymbol{\pi}_1)_p$ implies an occlusion ordering between these two objects, so that the first object can occlude the second one, but the opposite is not allowed.

In the general case with arbitrary number of objects the model (2.14) becomes

$$p(\mathbf{x}|j_b, j_1, \ldots, j_L) = \prod_{p=1}^{P} p(x_p|j_b, j_1, \ldots, j_L), \tag{2.15}$$

where $p(x_p|j_b, j_1, \ldots, j_L)$ is an $L+1$-component mixture model,

$$\begin{aligned} p(x_p|j_b, j_1, \ldots, j_L) &= \sum_{\ell=1}^{L} \prod_{k=1}^{\ell-1} (\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p (T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p; (T_{j_\ell}\mathbf{f}_\ell)_p) \\ &+ \prod_{k=1}^{L} (\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p p_b(x_p; (T_{j_b}\mathbf{b})_p), \end{aligned} \tag{2.16}$$

where if $\ell = 1$, then the term $\prod_{k=1}^{\ell-1}(\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p$ in equation (2.16) is defined to be equal to 1.

The order (from left to right) of the object models in equation (2.16) corresponds to the occlusion allowed between the objects. Particularly, the first object exists closest to the camera, thus it can never be occluded by any other object, the second object can only be occluded by the first object and so on. The background exists in the furthest distance from the camera.

Notice that in the above model there is an asymmetry between the objects because of the specified occlusion ordering. If the objects can arbitrarily occlude one another so that the occlusion ordering can change from image to image, then the above model is no longer appropriate. A principled way to deal with this situation is to consider all $L!$ possible rearrangements of the objects (using an additional hidden variable). See section 3.4 at chapter 3 for more details about this.

Learning the parameters of this model using an exact EM algorithm is intractable. This is because the E-step requires searching over $O(J_f^L J_b)$ configurations in order to express the posterior probability of each configuration given the image. Jojic and Frey (2001) have considered a variational approximation. In this thesis we have developed a greedy algorithm that can learn the objects one after the other using a robust statistical method. Chapter 3 is dedicated to describe this algorithm in detail so we will not further discuss it at the current chapter, except to mention that this greedy algorithm learns one object at each stage, so that we need to search only over the transformations of a single object.

2.3 Discussion

To summarize, in this chapter we presented a probabilistic generative model for learning multiple objects from a set of images. However, we did not present any algorithm for learning the model parameters as the next chapter is dedicated to present the greedy algorithm we have developed. The generative model is similar to the Jojic and Frey (2001) method who adopt a variational EM algorithm for learning. In section 3.5 we discuss the differences of our method with the Jojic and Frey (2001) approach.

The generative model described in this chapter for learning multiple objects is a step towards learning appearance based object models for object recognition using unsupervised learning. Of course, the above framework still is not suitable for object recognition of categories since it can only be used to learn specific rigid objects that move under 2D planar transformations. However, in chapter 4 we enhance this model so that to find rigid parts of articulated specific objects such as human body and learn a joint distribution that connects these parts. In addition, in chapter 6 we discuss several other improvements of the current model as well as a training scenario where our algorithm can be used as a preprocessing step of learning a recognition system for object categories using unsupervised learning.

Chapter 3

Greedy learning of multiple objects using robust statistics

Exact EM training of the generative model for multiple objects presented in the previous chapter is intractable. This is because as the number of objects increases, there is a combinatorial explosion of the number of configurations that need to be considered. The above problem is of general concern since it is related to the factorial learning problem where multiple causes (objects) are needed to explain the data (image). In this chapter we develop a greedy algorithm that extracts the object models sequentially from the data, thus avoiding the combinatorial explosion. The algorithm learns first the background while the foreground objects are found at later stages. This is achieved by making use of a robust statistical method so that when we learn an object all the areas in the image where the other objects exist are explained as outliers.

The structure of the remainder of this chapter is as follows. In section 3.1 we review the factorial learning problem. In section 3.2 we show how we can learn one foreground object against background using robust statistics. In section 3.3 we describe all the stages of the greedy algorithm. In section 3.4 we compute the occlusion ordering of the foreground objects in an image and we also describe how we can refine all the appearances of the objects by maximizing the complete data log likelihood over all the parameters. In 3.5 we discuss related work. In section 3.6 we present experimental results finding objects appearing against static, moving and random backgrounds and

we conclude with a discussion in section 3.7.

3.1 Factorial learning

As mentioned in section 2.2.3 in the previous chapter an exact EM algorithm for learning the parameters of the model for multiple objects is intractable. To get an intuition why this is the case, note that the generative model assumes that an image is produced by independently choosing the location (transformation) of each object and then instantiating all the objects within the image. These assumptions of the generative process can be represented as a directed graphical model depicted in Figure 3.1. Clearly, for J_b possible transformations of the background and J_f transformations of the L foreground objects, there are $J_f^L J_b$ ways to generate an image. This implies that a full search over all these $J_f^L J_b$ possibilities that the EM algorithm needs to evaluate in the E-step is infeasible.

The generative model for learning multiple objects is a factorial learning type of model. By factorial learning we mean a situation where multiple causes (factors) are needed to explain the observed data (image)[1] as illustrated in graphical model of Figure 3.1. The general problem of factorial learning has longer history, see, for example, Barlow (1989), Hinton and Zemel (1994), Saund (1995) and Ghahramani (1995). A serious concern with the factorial learning problem is that as the number of causes (objects) increases, there is always this combinatorial explosion of the number of configurations that need to be considered. Since learning using an exact EM algorithm is intractable approximations should be considered. Ghahramani (1995) suggests mean field and Gibbs sampling approximations and Jojic and Frey (2001) use approximate variational inference. Below we describe a different learning algorithm by finding the background and the foreground objects sequentially one after the other using a robust statistical method. Sequential object discovery is possible because multiple objects combine by occluding each other and/or the background.

In the next section we discuss how using robust statistics we can discover only one foreground object and ignore all the rest foreground objects that might appear in the

[1] This is the same terminology as used in the factor analysis model from statistics, although that model uses linear (Gaussian) assumptions.

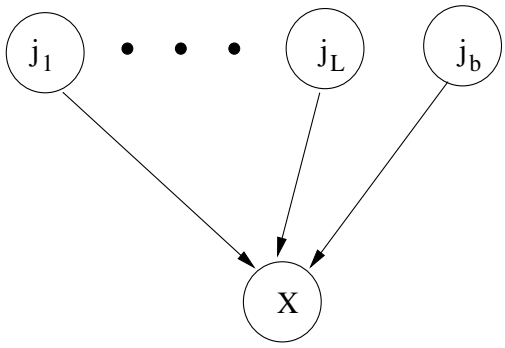


Figure 3.1: Graphical representation of the generative model for learning multiple objects.

images. This will motivate the use of the greedy algorithm presented in section 3.3.

3.2 Learning one object using robust statistics

Suppose a set of images showing multiple foreground objects against a background. Apart from only one foreground object being modelled we consider the rest of the foreground objects as "outlying" information and thus we do not wish to model their appearances. In other words, our objective is to learn only the one object of interest and efficiently "ignore" all the rest objects. A way to do this is to robustify the model described in section 2.2.2 so that foreground and background occlusion can be tolerated. More specifically, for a foreground pixel, some other objects may be interposed between the camera and our object, thus perturbing the pixel value. This can be modelled with a mixture distribution as

$$p_f(x_p; f_p) = \alpha_f N(x_p; f_p, \sigma_f^2) + (1 - \alpha_f) U(x_p), \tag{3.1}$$

where α_f is the fraction of times a foreground pixel is not occluded and the robustifying component $U(x_p)$ is a uniform distribution common for all image pixels. When a object pixel is occluded this should be explained by the uniform component. Such robust models have been used for image matching tasks by a number of authors, notably Black and colleagues (Black and Jepson, 1996).

Similarly for the background, a different object from the one being modelled may be interposed between the background and the camera, so that we again have a mixture model

$$p_b(x_p; b_p) = \alpha_b N(x_p; b_p, \sigma_b^2) + (1 - \alpha_b) U(x_p), \tag{3.2}$$

with similar semantics for the parameter α_b. Note that for random backgrounds the above robustification make less sense (since the Gaussian will have large variance σ_b^2) but it will apply to the static or moving background cases.

Now the probability model for the image $\mathbf{x}$ given the transformation of the foreground object is given as in equation (2.9), with the only difference that the p_f and p_b densities are not Gaussians any more but are robustified as explained above. Assuming a static background, training this model is completely analogous to the non-robust case (see section 2.2.2 and appendix A.2), for example applying the EM has complexity $O(J_f)$.

Note that it is not necessary that the robustifying component be a uniform distribution, for example a broad Gaussian would also work. However, as pixels do have maximum and minimum values the uniform distribution is a natural choice, and is also the maximum entropy distribution.

In practice the above framework using robust statistics can be used to learn multiple objects in images. By random parameter initializations, and on different runs we can find different objects. We denote such an algorithm as random starts. However, we have found (Williams and Titsias, 2003) that this is rather inefficient as the basins of attraction for the different objects may be very different in size given the initialization. For example, this algorithm will tend always to discover the largest or dominant object so it will be unlikely to find small objects. Next section describes a greedy algorithm that can discover the objects sequentially.

3.3 Greedy learning of multiple objects

In this section we describe a greedy algorithm for training the generative model for multiple objects. This algorithm optimizes the parameters of the model by stages so that at each stage an object is discovered.

The model for L foreground objects and a moving background assumes that a image is generated by first choosing values for the transformations $(j_b, j_1, \ldots, j_L)$ and then drawing $\mathbf{x}$ according to[2]

$$p(\mathbf{x}|j_b, j_1, \ldots, j_L) = \prod_{p=1}^{P} p(x_p|j_b, j_1, \ldots, j_L), \tag{3.3}$$

where $p(x_p|j_b, j_1, \ldots, j_L)$ is the $L+1$-component mixture model

$$\begin{aligned} p(x_p|j_b, j_1, \ldots, j_L) &= \sum_{\ell=1}^{L} \prod_{k=1}^{\ell-1} (\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p (T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p; (T_{j_\ell}\mathbf{f}_\ell)_p) \\ &+ \prod_{k=1}^{L} (\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p p_b(x_p; (T_{j_b}\mathbf{b})_p), \end{aligned} \tag{3.4}$$

where if $\ell = 1$ then the term $\prod_{k=1}^{\ell-1}(\mathbf{1} - T_{j_k}\boldsymbol{\pi}_k)_p$ in (3.4) is defined to be equal to 1.

The greedy algorithm is based on using robust statistics similarly to what explained in the previous section. Particularly, from now on we will assume that both the foreground p_{f_ℓ} and background p_b pixels densities are robustified as described in section 3.2. This robustification is the key for our greedy algorithm to find the objects one at a time.

Each component in the mixture distribution of equation (3.4) corresponds to an object model which is either one of the L foreground objects or the background. The key idea of our learning algorithm is to learn this mixture model (and thus the relation with the associated transformation variable) sequentially, by learning the objects one at a time. An intuitive way to introduce this algorithm is that originally we constrain the mixture distribution so that the background takes all the probability and the masks of the foreground objects are zero. Since the background pixel densities are robustified according to equation (3.2) we can learn the background by "ignoring" all the foreground objects. When a pixel of the background is occluded by a foreground object that should be explained by the outlier component in (3.2), so that the pixel will not affect the estimation of the background. At each subsequent stage the mask of a foreground object is set free to get a non zero value and the corresponding object model is learned.

[2]For clarity we repeat here some equations that are also given in section 2.2.3, where we introduced the model.

Below we first describe learning the background in section 3.3.1, then discuss learning the first object in section 3.3.2 and further objects in section 3.3.3. We summarize the algorithm in section 3.3.4. Further details are given the appendix B.

3.3.1 Finding the background

The greedy algorithm starts by first finding the background. By constraining all the masks $\{\boldsymbol{\pi}_\ell\}_{\ell=1}^{L}$ to be zero, the mixture model (3.4) has only one component (corresponding to the background) and thus equation (3.3) takes the form

$$p(\mathbf{x}|j_b) = \prod_{p=1}^{P} p_b(x_p; (T_{j_b}\mathbf{b})_p), \tag{3.5}$$

where p_b is robustified according to equation (3.2). Assuming an uniform prior P_{j_b} for transformation j_b the log likelihood of the training images is

$$L_b = \sum_{n=1}^{N} \log \sum_{j_b=1}^{J} P_{j_b} p(\mathbf{x}^n|j_b), \tag{3.6}$$

which can be maximized with respect to $\{\mathbf{b}, \sigma_b^2\}$ by the EM algorithm. This algorithm searches over J_b possibilities and is tractable; details are provided in the appendix B.1.

3.3.2 Finding the first object

At the second stage of the algorithm we learn the first foreground object. By allowing the mask $\boldsymbol{\pi}_1$ to take on non-zero values equation (3.3) becomes

$$\begin{aligned} p(\mathbf{x}|j_b, j_1) &= \prod_{p=1}^{P} (T_{j_1}\boldsymbol{\pi}_1)_p p_{f_1}(x_p; (T_{j_1}\mathbf{f}_1)_p) + (\mathbf{1} - T_{j_1}\boldsymbol{\pi}_1)_p p_b(x_p; (T_{j_b}\mathbf{b})_p) \\ &= \prod_{p=1}^{P} p(x_p|j_b, j_1). \end{aligned} \tag{3.7}$$

The log likelihood of the training images is $L_1 = \sum_{n=1}^{N} \log \sum_{j_1, j_b} P_{j_1} P_{j_b} p(\mathbf{x}^n|j_1, j_b)$ and a direct maximization using the EM algorithm can be quite demanding, since inference involves searching over $J_f J_b$ possibilities. Our greedy algorithm drops the complexity of the search to J_f possibilities by applying a constrained EM algorithm (Neal and Hinton, 1998) that exploits the fact that we already know the background. In particular, for

each training image $\mathbf{x}^n$ we introduce the distribution $Q^n(j_1, j_b) = Q^n(j_1|j_b)Q^n(j_b)$ over the transformations and we express a lower bound (based on the Jensen's inequality) of the log likelihood L_1:

$$F_1 = \sum_{n=1}^{N} \sum_{j_b, j_1} Q^n(j_1|j_b)Q^n(j_b) \left\{ \log \left(P_{j_1} P_{j_b} \prod_{p=1}^{P} p(x_p^n|j_b, j_1) \right) - \log Q^n(j_1|j_b)Q^n(j_b) \right\}. \quad (3.8)$$

This lower bound becomes tight by choosing $Q^n(j_b, j_1)$ to be the posterior $P(j_b, j_1|\mathbf{x}^n)$ for every image $\mathbf{x}^n$. Since we have learned the background we can use the posterior probability $P(j_b|\mathbf{x}^n)$ (computed as described in appendix B.1) to find the most probable transformation j_b^n that best explains image $\mathbf{x}^n$ and then we approximate $Q^n(j_b)$ so that it gives probability one for $j_b = j_b^n$ and zero for the remaining values[3]. Thus, F_1 takes the form

$$F_1 = \sum_{n=1}^{N} \sum_{j_1=1}^{J_f} Q^n(j_1) \left\{ \sum_{p=1}^{P} \log p(x_p^n|j_b^n, j_1) - \log Q^n(j_1) \right\} + const, \quad (3.9)$$

where *const* depends on the uniform probabilities P_{j_b} and P_{j_1}. Also the dependence of $Q^n(j_1)$ on j_b^n for simplicity has been omitted from our notation.

Maximization of F_1 can be carried by the EM algorithm, where in the E-step we maximize F_1 with respect to the Q^n distributions (see equation (B.7)) and in the M-step with respect to the object parameters $\{\mathbf{f}_1, \boldsymbol{\pi}_1, \sigma_1^2\}$. Thus, the computational complexity for learning the object has been kept to a minimum since we have only to search over the J_f possible transformations of the object. Recall that the pixel densities p_{f_1} and p_b are robustified which allows us to deal with occlusion that can be caused by the remaining $L-1$ not-yet-discovered objects.

3.3.3 Learning further objects

The algorithm for learning the second and subsequent foreground objects is a bit more complicated as we subtract out the objects learned so far. We first describe how we

[3]It would be possible to make a "softer" version of this, where the transformations are weighted by their posterior probabilities, but in practice we have found that these probabilities are usually 1 for the best-fitting transformation and 0 otherwise after learning.

learn the second object and then generalize to the case of the ℓth object.

To learn a second foreground object we first allow the mask $\boldsymbol{\pi}_2$ to take on non-zero values so that the conditional density of $\mathbf{x}$ given the hidden transformations becomes $P(\mathbf{x}|j_b, j_1, j_2) = \prod_{p=1}^{P} p(x_p|j_b, j_1, j_2)$ where $p(x_p|j_b, j_1, j_2)$ is given by equation (3.4). We learn the second object by maximizing a lower bound of the log likelihood $L_2 = \sum_{n=1}^{N} \log \sum_{j_b, j_1, j_2} P_{j_b} P_{j_1} P_{j_2} p(\mathbf{x}^n|j_b, j_1, j_2)$. Particularly, since we have learned the background and the first object we use the most probable transformations j_b^n and j_1^n that explain image $\mathbf{x}^n$ to lower bound the log likelihood L_2:

$$F_2 = \sum_{n=1}^{N} \sum_{j_2=1}^{J_f} Q^n(j_2) \left\{ \sum_{p=1}^{P} \log p(x_p^n|j_b^n, j_1^n, j_2) - \log Q^n(j_2) \right\} + const. \tag{3.10}$$

F_2 can be tractably optimized over $Q^n(j_2)$ and over the parameters of the second object $\{\mathbf{f}_2, \boldsymbol{\pi}_2, \sigma_2\}$. However, we can make the search for the second object much more efficient (ensuring that we will find a different object) by further constraining equation (3.10) so that all the pixels belonging to the first object are removed from consideration[4]. First of all note that the values of the transformed mask $T_{j_1^n}\boldsymbol{\pi}_1$ will be close to 1 for all pixels of image $\mathbf{x}^n$ that are part of the first object. All these pixels should be removed from consideration unless there are occluded by other not-yet-discovered objects. Thus, we consider the vector $\boldsymbol{\rho}_1^n = (T_{j_1^n}\boldsymbol{\pi}_1) * \mathbf{r}_{j_1^n}^n$ where $\mathbf{y} * \mathbf{z}$ denotes the element-wise product of the vectors $\mathbf{y}$ and $\mathbf{z}$ and $(\mathbf{r}_{j_1^n}^n)_p = \frac{\alpha_f N(x_p^n; (T_{j_1^n}\mathbf{f}_1)_p, \sigma_1^2)}{\alpha_f N(x_p^n; (T_{j_1^n}\mathbf{f}_1)_p, \sigma_1^2) + (1-\alpha_f) U(x_p^n)}$. Thus, $\boldsymbol{\rho}_1^n$ will roughly give values close to 1 only for the non-occluded object pixels of image $\mathbf{x}^n$, and these are the pixels that we wish to remove from consideration. Now considering $(\boldsymbol{\rho}_1^n)_p$ as the probability according to which the pixel p of image $\mathbf{x}^n$ is part of the first object we once again lower bound F_2 using the inequality

[4]Notice that we have implemented an alternative algorithm which directly maximizes (3.10) with respect to the parameters of the second object. However, this algorithm in some experiments was problematic as it found the same object twice. On the other hand the algorithm that removes from consideration all the pixels of the first object in each training image was always successful in discovering a different object.

$\log\sum_i y_i = \log(\sum_i \frac{y_i}{p_i} p_i) \geq \sum_i p_i \log\frac{y_i}{p_i}$ (obtained from Jensen's inequality) to obtain

$$\begin{aligned} F_2 &= \sum_{n=1}^{N}\sum_{j_2=1}^{J_f} Q^n(j_2)\Bigg\{\sum_{p=1}^{P}(\boldsymbol{\rho}_1^n)_p \log\Big\{(T_{j_1^n}\boldsymbol{\pi}_1)_p p_{f_1}(x_p^n;(T_{j_1^n}\mathbf{f}_1)_p)\Big\} + (\overline{\boldsymbol{\rho}}_1^n)_p \times \\ &\quad \log\Big\{(\mathbf{1}-T_{j_1^n}\boldsymbol{\pi}_1)_p[(T_{j_2}\boldsymbol{\pi}_2)_p p_{f_2}(x_p^n;(T_{j_2}\mathbf{f}_2)_p) + (\mathbf{1}-T_{j_2}\boldsymbol{\pi}_2)_p p_b(x_p^n;(T_{j_b}^n\mathbf{b})_p)]\Big\} \\ &\quad - \log Q^n(j_2)\Bigg\} + const, \end{aligned} \tag{3.11}$$

where $\overline{\boldsymbol{\rho}}_1^n = \mathbf{1} - \boldsymbol{\rho}_1^n$ and the *const* is a constant term containing the entropic term $-\sum_{n=1}^{N}\sum_{p=1}^{P}\left\{(\boldsymbol{\rho}_1^n)_p \log(\boldsymbol{\rho}_1^n)_p + (\overline{\boldsymbol{\rho}}_1^n)_p \log(\overline{\boldsymbol{\rho}}_1^n)_p\right\}$ plus terms involving the uniform probabilities $\{P_{j_b}, P_{j_1}, P_{j_2}\}$. Since the parameters of the first object are fixed, the above quantity is further written as

$$\begin{aligned} F_2 = \sum_{n=1}^{N}\sum_{j_2=1}^{J_f} Q^n(j_2)\Big\{\sum_{p=1}^{P}(\overline{\boldsymbol{\rho}}_1^n)_p \log[(T_{j_2}\boldsymbol{\pi}_2)_p p_{f_2}(x_p^n;(T_{j_2}\mathbf{f}_2)_p) + \\ (\mathbf{1}-T_{j_2}\boldsymbol{\pi}_2)_p p_b(x_p^n;(T_{j_b^n}\mathbf{b})_p)] - \log Q^n(j_2)\Big\} + const. \end{aligned} \tag{3.12}$$

Note that when for a pixel p of image $\mathbf{x}^n$ $(\overline{\boldsymbol{\rho}}_1^n)_p \simeq 0$, this pixel is removed from consideration (in a probabilistic fashion) according to equation (3.12).

Further objects are learned similarly to the two-objects case except that the pixels of all previously learned foreground objects should be removed from consideration. This is achieved by setting $\mathbf{z}_0^n = \mathbf{1}$ for all $n = 1,\ldots,N$ and using the recursion $\mathbf{z}_\ell^n = \mathbf{z}_{\ell-1}^n * \overline{\boldsymbol{\rho}}_\ell^n$. Note that $\mathbf{z}_1^n = \overline{\boldsymbol{\rho}}_1^n$. For object ℓ the objective function F_ℓ given in equation (3.13) is optimized to yield $\{\mathbf{f}_\ell, \boldsymbol{\pi}_\ell$ and $\sigma_\ell^2\}$.

Note that the greedy algorithm treats the background and the rest L objects differently, since pixels ascribed to the non-occluded background are not removed from consideration as is the case for the foreground objects. We implemented an alternative greedy algorithm that treats the background similarly to the remaining objects (removing non-occluded background pixels from consideration). However, this algorithm did not work so well in practice as some pixels can wrongly be removed from consideration because their values happen to agree with pixels of the occluding object. For the background the number of such pixels can be large since the background is always occluded by all of the L objects. We observed experimentally that this can result in

noisy estimates for some of the L foreground objects since many of their pixels are accidentally removed from consideration after the background is learned. On the other hand in the case of the foreground objects is not problematic since occlusion occurs only in some images and is typically partial.

3.3.4 Summary of the greedy algorithm

1. Learn the background and infer the most probable transformation j_b^n for each image $\mathbf{x}^n$.

2. Initialize the vectors $\mathbf{z}_0^n = \mathbf{1}$ for $n = 1, \ldots, N$.

3. For $\ell = 1$ to L

   (a) Learn the ℓth object parameters $\{\mathbf{f}_\ell, \boldsymbol{\pi}_\ell, \sigma_\ell^2\}$ by maximizing F_ℓ using EM algorithm, where

$$F_\ell = \sum_{n=1}^{N} \sum_{j_\ell=1}^{J_f} Q^n(j_\ell) \Big\{ \sum_{p=1}^{P} (\mathbf{z}_{\ell-1}^n)_p \log[(T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p; (T_{j_\ell}\mathbf{f}_\ell)_p) + (\mathbf{1} - T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_b(x_p; (T_{j_b^n}\mathbf{b})_p)] - \log Q^n(j_\ell) \Big\}. \quad (3.13)$$

   (b) Infer the most probable transformation $\{j_\ell^n\}$ and update the weights $\mathbf{z}_\ell^n = \mathbf{z}_{\ell-1}^n * \overline{\boldsymbol{\rho}}_\ell^n$ where $\overline{\boldsymbol{\rho}}_\ell^n$ is computed as described in the text.

The update equations used at any stage of the above algorithm are given in appendix B.

3.4 Specification of the occlusion ordering and refinement of the object models

Once we run the greedy algorithm, we obtain a set of appearance models for the foreground objects and the background as well as an approximation of the transformation of each object in each training image. To better specify the generation process of each image we have to compute the occlusion ordering of the foreground objects. This

is necessary since even when the occlusion ordering remains fixed across all training images, the greedy algorithm might not pull out the objects in accordance with this ordering, i.e. discovering first the closest from the camera object, then the second closest from the camera object etc. The order the greedy algorithm finds the objects strongly depends on the initialization of the parameters and the size of the objects[5]. For example, for the data of Figure 3.2a the greedy algorithm in some runs discovers first the person with the white shirt who in some frames is occluded. Below we discuss how we compute the occlusion ordering.

A way to infer the occlusion ordering of the foreground objects or layers in an image $\mathbf{x}^n$ is to consider all possible permutations of these layers and choose the permutation that gives the maximum likelihood. The simplest case is to have two foreground objects with parameters $(\boldsymbol{\pi}_1, \mathbf{f}_1)$ and $(\boldsymbol{\pi}_2, \mathbf{f}_2)$, respectively. Given the approximated transformations, the log likelihood values of the two possible orderings are

$$\begin{aligned} L_{12}^n &= \sum_{p=1}^{P} \log\Big\{ (T_{j_1^n}\boldsymbol{\pi}_1)_p p_{f_1}(x_p^n; (T_{j_1^n}\mathbf{f}_1)_p) + (\mathbf{1} - T_{j_1^n}\boldsymbol{\pi}_1)_p \times \\ &\quad [(T_{j_2^n}\boldsymbol{\pi}_2)_p p_{f_2}(x_p^n; (T_{j_2^n}\mathbf{f}_2)_p) + (\mathbf{1} - T_{j_2^n}\boldsymbol{\pi}_2)_p p_b(x_p^n; (T_{j_b^n}\mathbf{b})_p)] \Big\}, \quad (3.14) \end{aligned}$$

and

$$\begin{aligned} L_{21}^n &= \sum_{p=1}^{P} \log\Big\{ (T_{j_2^n}\boldsymbol{\pi}_2)_p p_{f_2}(x_p^n; (T_{j_2^n}\mathbf{f}_2)_p) + (\mathbf{1} - T_{j_2^n}\boldsymbol{\pi}_2)_p \times \\ &\quad [(T_{j_1^n}\boldsymbol{\pi}_1)_p p_{f_1}(x_p^n; (T_{j_1^n}\mathbf{f}_1)_p) + (\mathbf{1} - T_{j_1^n}\boldsymbol{\pi}_1)_p p_b(x_p^n; (T_{j_b^n}\mathbf{b})_p)] \Big\}. \quad (3.15) \end{aligned}$$

The selected occlusion ordering for the image $\mathbf{x}^n$ is the one with the largest log likelihood, i.e. $L_{kl}^n > L_{lk}^n$. If we know a priori that the occlusion ordering across all images remains fixed, which often happens in video data, then we select the ordering with the largest overall log likelihood $L_{kl} = \sum_{n=1}^{N} L_{kl}^n$. When we have L foreground objects we work exactly analogously as above by expressing all $L!$ permutations of the foreground layers and select the one with the largest likelihood.

The above computation of the occlusion ordering takes $L!$ time and it can be used only when we have few foreground objects (less than 7). However, in most of the cases we can further speed up this computation and estimate the occlusion ordering for large

[5] Objects that occupy more pixels than others are more likely to be found first.

number of objects. The idea is that an object ℓ usually does not overlap (either occludes or is occluded) with all the rest $L-1$ objects, but only with some of them. Thus, if for each object we identify the overlapping objects, the complexity in the worse case will be $O(G!)$ where G is the largest number of objects that simultaneously overlap with each other. Details of this algorithm together with illustrative examples are given in appendix B.3.

Once the occlusion ordering has been specified for each training image, we can maximize the complete (using the approximated transformations and the occlusion orderings) log likelihood for the model described by equations (2.15) and (2.16) and refine the appearances and masks of the objects. Note that for this maximization we need the EM algorithm in order to deal with the fact that each pixel follows a $L+1$-component mixture distribution (see equation (2.16)). However, this EM runs quickly since all the transformations have been "filled in" with the approximated values provided by the greedy algorithm, thus there is no search over the transformations of the objects.

3.5 Related work

Jojic and Frey (2001) presented a similar generative model for learning multiple objects and use a variational Generalized EM algorithm for learning. In their model the different layers are synthesized according to an alpha matting (or pixel mixing) rule that allows transparency[6]. In contrast, in our model the layers are synthesized through a mixture distribution (see e.g. equation (2.9)) which ensures that the foreground and background appearances strictly combine by occlusion, thus no transparency between them is allowed. When the objects combine by occlusion, our model allows for a more robust inference and learning. For example, an exact M-step of the EM algorithm can be expressed as opposed to the generalized M-step used by Jojic and Frey. This fact has also been used by Kannan et al. (2005) and Frey and Jojic (2004). In terms of the learning algorithms the variational EM of Jojic and Frey (2001) updates all the param-

[6]For example, for one foreground object against a background an image is drawn by a Gaussian with mean $\mathbf{m} * \mathbf{f} + (\mathbf{1} - \mathbf{m}) * \mathbf{b}$ where $\mathbf{m}$ is the transparency mask that takes values in the range $[0,1]$ and $*$ stands for the element-wise product between vectors.

eters of the model simultaneously. This might suffer from local maxima because of the large amount of parameters that need to be initialized and optimized at the same time. The greedy algorithm learns one object at each stage and its initialization is straightforward since it will make no difference if we initialize the object parameters $(\boldsymbol{\pi}_\ell, \mathbf{f}_\ell)$ to the same point at each time[7]. Concerning speed, the complexity of the variational Generalized EM is $O(P(LJ_f + J_b))$ per training image while for the greedy is $O(\hat{P}(LJ_f + J_b))$, where $\hat{P}$ is the average number of pixels that are on (not removed from consideration) at each stage. The greedy algorithm can be slighter faster since $\hat{P} < P$. Finally, the greedy algorithm can be applied when the occlusion ordering of the foreground objects can change in different images (see e.g. the Experiments 4 and 5), while the method of Jojic and Frey (2001) assumes that the occlusion ordering is fixed across all images.

If video sequence data is available then it is possible to compute optical flow information, and this can be used as a cue to discover objects by clustering flow vectors into "layers". Some early work on this topic is by Wang and Adelson (1994), and an example of more recent work is that of Tao et al. (2000). Note that our method does not require a video sequence and can be applied to unordered collections of images, as illustrated in Experiments 4 and 5. Chapter 4 discusses learning from video data and section 4.4 gives more related work on learning objects from video.

In our work the model for each pixel is a mixture of Gaussians. There is some previous work on pixelwise mixtures of Gaussians (see, e.g. Rowe and Blake 1995) which can, for example, be used to achieve background subtraction and highlight moving objects against a static background. Our work extends beyond this by gathering the foreground pixels into objects, and also allows us to learn objects in the more difficult non-static background case.

The greedy method has an analogue in neural network methods for Principal Components Analysis (PCA). To carry out PCA we can extract the principal component using Hebbian learning. If we then subtract of the projection of the input onto the principal direction, we can again use Hebbian learning to extract the second principal component, and so on (Sanger, 1989). This process parallels the successive discov-

[7] Despite the fact that the initial parameter values can be the same, the greedy algorithm at each stage is forced to find a different object since the pixel of all previously discovered objects have been removed.

ery of objects in our method. However, notice that what is subtracted out from each input (image) is a global information and not specific pixels. We note also that this sequential algorithm cannot be used if a full factor analysis model (with different noise variances on different visible dimensions) is to be learned.

3.6 Experiments

We describe five experiments extracting movable objects from images using static, moving and random backgrounds. In these experiments the uniform distribution $U(x_p)$ is based on the maximum and minimum pixel values of all training image pixels. In all the experiments reported below except experiment 5 the parameters α_f and α_b were chosen[8] to be 0.9; in experiment 5 α_b was set to 1. Also we assume that the total number of objects L that appear in the images is known, thus the greedy algorithm terminates when we discover the Lth object.

To apply the greedy algorithm we have to initialize the model parameters at each stage. We first describe how we initialize the background parameters as the background is learned at the first stage of the algorithm. The background appearance $\mathbf{b}$ corresponds to an image that is larger than the input image size. We initialize the centred $P_x \times P_y$ block of $\mathbf{b}$ to be equal with the mean of the training images. The rest pixels of $\mathbf{b}$ are initialized by repeating the border-lines of pixels in the centred block of $\mathbf{b}$ and then adding a Gaussian noise to these pixels. The variance σ_b^2 is initialized to a large value (much larger value than the overall variance of all image pixels[9]). The parameters of an object learned at each subsequent stage of the greedy algorithm are always initialized in the same way. Each element of the mask $\boldsymbol{\pi}_\ell$ is initialized to 0.5, and the variance σ_ℓ^2 to a large value equal to the σ_b^2 initial value. To initialize the foreground appearance $\mathbf{f}_\ell$, we compute the pixelwise mean of the training images and add independent Gaussian noise with equal variances at each pixel, where the variance is set to be large enough so that the range of pixel values found in the training images can be explored.

[8] These parameters could be learned with some additional care but in our current implementation we do not do so.

[9] In our experiments the input image pixels are normalized to lie in $[0, 1]$ and the background variance σ_b^2 as well as the any foreground object variance σ_ℓ^2 is initialized to 2.

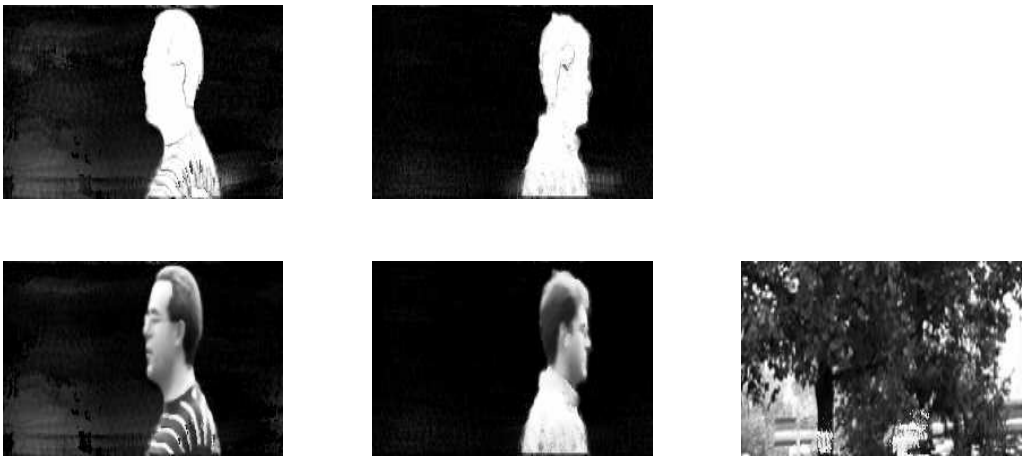
In all of the experiments described below the above initialization scheme proved to be effective and we obtained good results by performing one or two runs of the greedy algorithm. At each stage of the algorithm typically 100 iterations were sufficient to reach convergence.

Experiment 1. Figure 3.2 illustrates the detection of two objects against a static background[10]. Some examples of the 44 118×248 training images (excluding the black border) are shown in Figure 3.2a and results are shown for the greedy algorithm in Figure 3.2b. For both objects we show both the learned mask and the element-wise product of the learned foreground and mask. In most runs the person with the striped shirt (Frey) is discovered first. It is interesting to comment on how the greedy algorithm operates in case the person with the lighter shirt (Jojic) is found first. As explained in section 3.3, once an object is discovered by the greedy algorithm, roughly speaking its non-occluded pixels are removed from consideration. Figure 3.3 illustrates this point for two frames of the video sequence; one without occlusion and one with occlusion. Figure 3.3a shows the two video frames and Figure 3.3b the pixels (displayed in white) that are masked out from the next run. Note that when Frey occludes Jojic the white stripes of Frey's shirt are accounted for by the Jojic model. This is because the white colour of these stripes agrees with the learned white colour of Jojic's shirt. This does not cause problems for learning the second object (Frey) as there are many frames where the occlusion does not take place. In other experiments with two people wearing different coloured clothes no such effect takes place. Video sequences of the raw data and the extracted objects can be viewed at `http://www.dai.ed.ac.uk/homes/s0129556/lmo.html`.

Experiment 2. We also conducted an interesting variant on the above experiment. Rather than walking independently, two people now move together, keeping the same distance apart. This led to the extraction of a mask containing both people. Note that this is expected, since the pixels of the two people can be explained by the same transformation, so are considered as one object. Of course, it is open to debate whether we would wish to think of what is learned as one or two objects. In our opinion the ability to extract such regularities is very sensible, and quite widespread, e.g. in

[10]These data are used in Jojic and Frey (2001). We thank N. Jojic and B. Frey for making available these data via `http://www.psi.toronto.edu/layers.html`.

(a)



(b)

Figure 3.2: Learning two objects against a static background. Panel (a) displays some frames of the training images, and (b) shows the two objects and background found by the greedy algorithm. The plots in the upper row of (b) show the masks $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$. The first two plots in the lower row of (b) display the element-wise products $\boldsymbol{\pi}_1 * \mathbf{f}_1$ and $\boldsymbol{\pi}_2 * \mathbf{f}_2$ while the third plot displays the background $\mathbf{b}$.

(a) (b)

Figure 3.3: What the greedy algorithm removes from consideration once Jojic is found. Panel (a) displays two frames of the training images, and (b) plots the corresponding $\boldsymbol{\rho}_1$ vectors (see section 3.3), that indicate the pixels which are masked out from the second iteration.

finding pairs of eyes. If it was desired, it would be a simple matter to run a connected components algorithm on the thresholded mask to pick out the two objects.

Experiment 3. In the data shown in Figure 3.4 two objects move against a moving background. Figure 3.4a shows some of the 36 70×140 images of the video sequence. Note that the background changes from frame to frame because of the camera's movement. Notice also that there is motion blur in some of the frames and that one person is occluded by the other in many frames as they walk in the same direction. Figure 3.4b shows the results of the greedy algorithm where at the first stage we find the background and at the next two stages the moving objects are found.

Experiment 4. In Figure 3.5 five objects are learned against a static background, using a dataset of 80 images of size 66×88. Notice the large amount of occlusion in some of the training images shown in Figure 3.5a. Results are shown in Figure 3.5b for the greedy algorithm.

Experiment 5. In Figure 3.6 we consider learning objects against random backgrounds. Actually three different backgrounds were used, as can be seen in the example

Figure 3.4: Learning two objects against a moving background. Panel (a) displays some frames of the training images, and (b) the panorama-background and the masks and rendered objects found by the greedy algorithm. (To show the rendered objects we reverse our usual convention and show the objects against a light background as the objects themselves are mainly dark.)

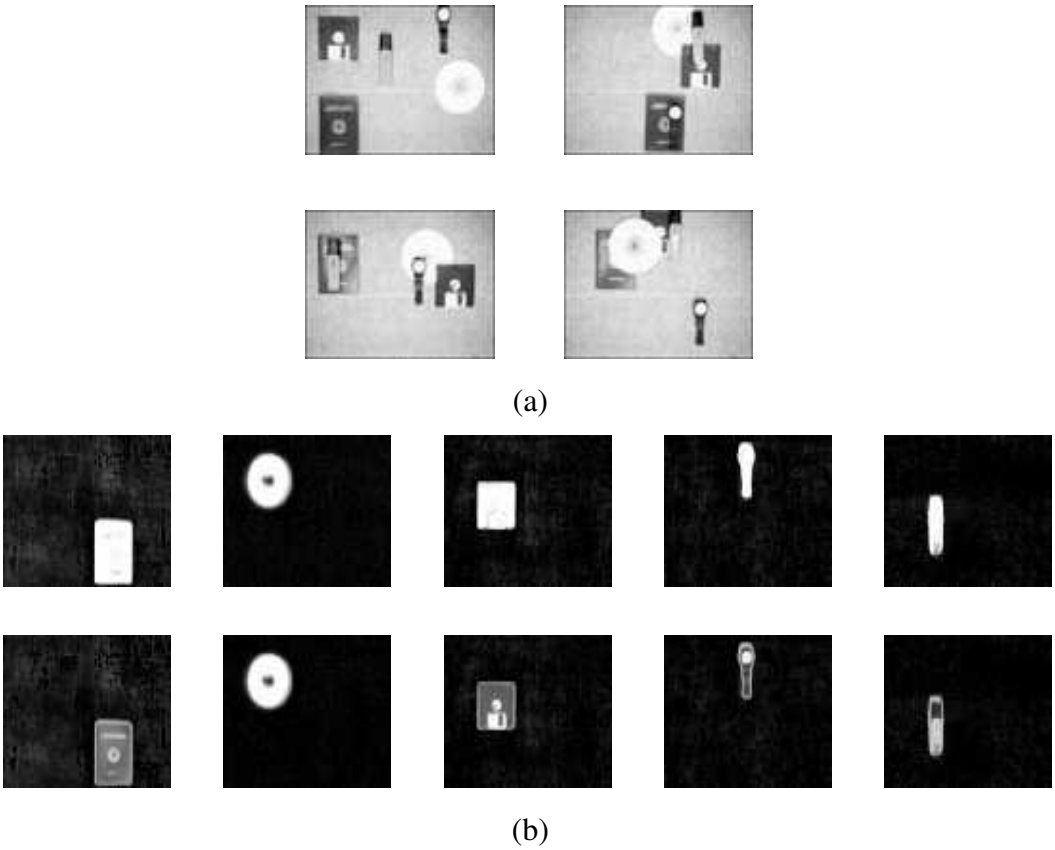
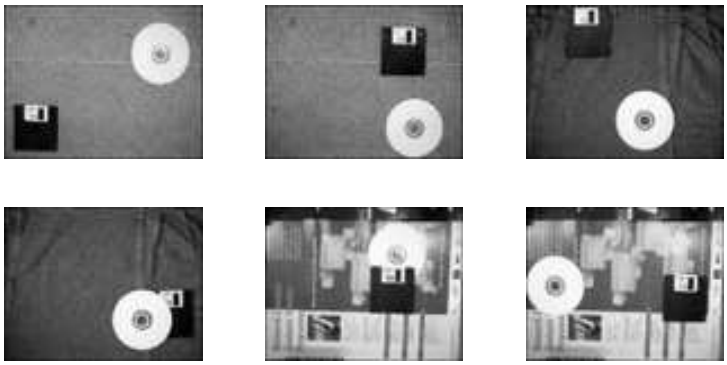
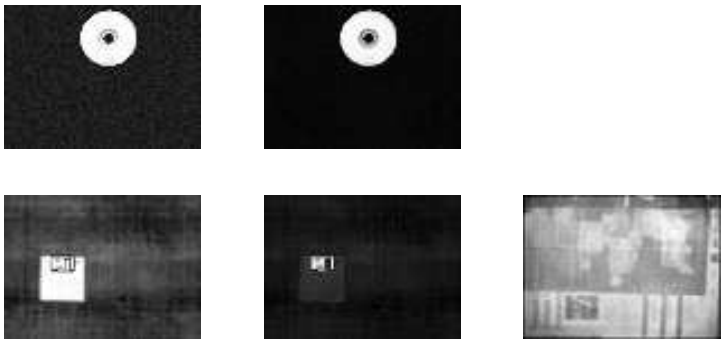


Figure 3.5: Learning five objects against a static background. Panel (a) displays some of the training images and (b) shows the masks and objects (displayed as described in the caption of Figure 3.2) learned by the greedy algorithm.



(a)



(b)

Figure 3.6: Two objects are learned from a set of images with three different backgrounds. Panel (a) displays some examples of the training images, and (b) shows the masks and objects found by the greedy algorithm, displayed as described in the caption of Figure 3.2.

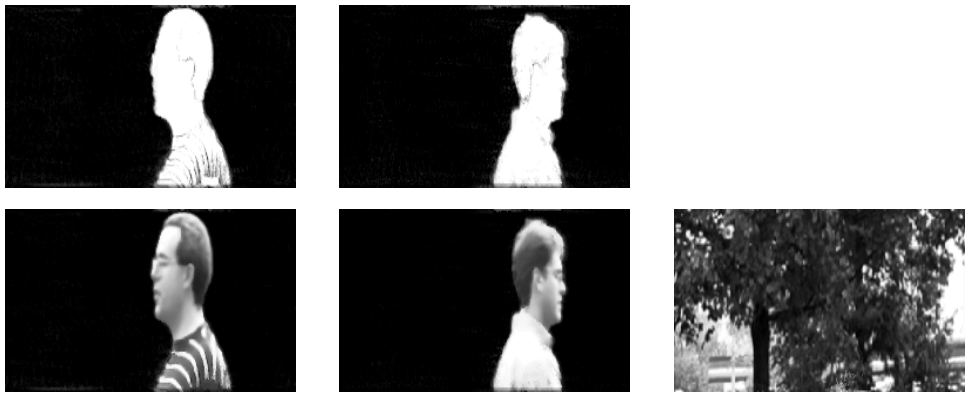


Figure 3.7: The appearances of the objects after the refinement step for the images used in Experiment 1. Comparing the above plots with the corresponding plots showing in Figure 3.2b, we can inspect the improvement over the parameters of the objects.

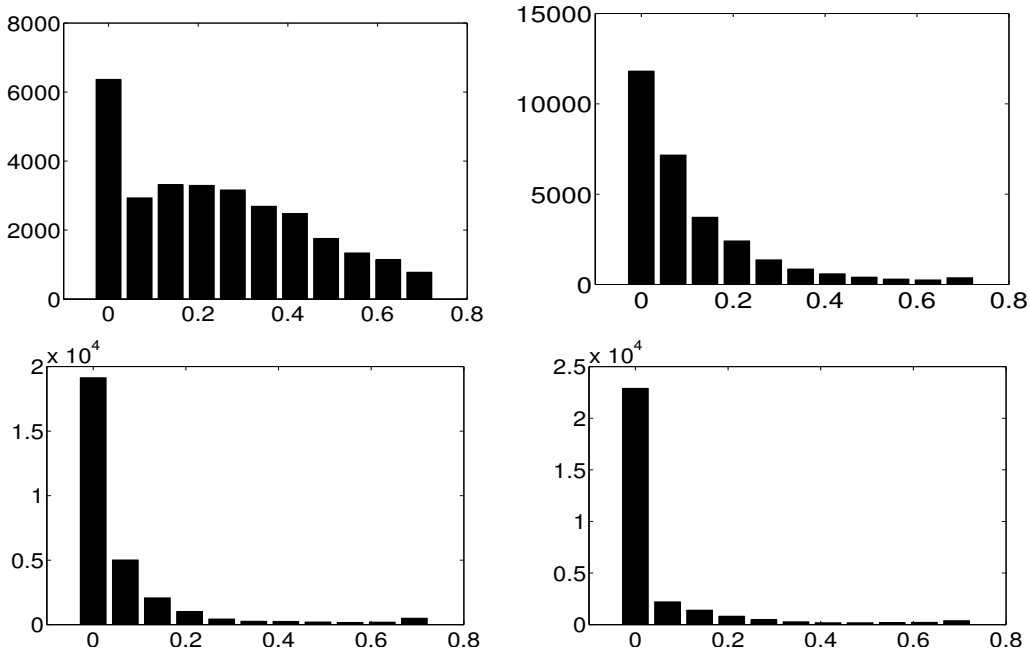


Figure 3.8: The two panels of the top row show the histograms of the entropy values for the masks of Figure 3.2b (the histogram on the left corresponds to Frey and the histogram on the right to Jojic). The bottom row shows the corresponding histograms (again the left plot corresponds to Frey and the right to Jojic) for the entropy values of the masks after the refinement step.

images shown in Figure 3.6a. Note that in this case we set $\alpha_b = 1$ since robustifying a random background does not make sense. There were 67 66×88 images in the training set. The results with the greedy algorithm are shown in Figure 3.6b.

In some other experiments using a few random backgrounds our algorithm has not worked well. In these cases it seems that the foreground models tend to model structure that appears in some backgrounds rather than the foreground objects. These problems might be overcome by using more random backgrounds, as this would fit the random background assumptions better.

Demonstration of the refinement of the appearances of the objects

In all the above experiments the occlusion ordering of the objects has been correctly computed according to the algorithm described in section 3.4 and appendix B.3. As explained in section 3.4, once the occlusion ordering has been specified for each training image we can refine the appearances of the objects by maximizing the complete data log likelihood over all the parameters. For this refinement step, we initialize the background and the foreground appearances and masks using the values provided by the greedy algorithm. The variances $(\sigma_1, \ldots, \sigma_L, \sigma_b)$ are reinitialized to a large value (as described in the second paragraph of this section) so that to escape from local maxima. Note also that for this maximization we maintain the robustification of the background and foreground pixel densities (α_b and α_f obtains the value 0.9) in order to deal with a possible deformability of the objects, e.g. local clothing deformation, or changes of the lighting conditions. The EM algorithms needed for the above maximization converges in few iterations (e.g. less than 20). This is because both the objects' appearances initialized by the greedy algorithm are already close to their final values and all the transformations of the objects have been "filled in" with the approximated values provided by the greedy algorithm.

Figure 3.7 displays the results for the Frey-Jojic images used in the Experiment 1 after the refinement step. Comparing Figure 3.2b with Figure 3.7 we can visually inspect the improvement over the appearances of the objects, e.g. the ghosts in the masks of Figure 3.2b have disappeared in Figure 3.7. To better examine the improvement in the mask $\boldsymbol{\pi}_\ell$ of an object, we measure the entropy in each pixel of the mask accord-

ing to $H\left((\boldsymbol{\pi}_\ell)_p\right) = -(\boldsymbol{\pi}_\ell)_p \log(\boldsymbol{\pi}_\ell)_p - (\mathbf{1}-\boldsymbol{\pi}_\ell)_p \log(\mathbf{1}-\boldsymbol{\pi}_\ell)_p$. Typically, a good mask has values close to 0 and 1 which can be indicated by the entropy values i.e. in such case $H\left((\boldsymbol{\pi}_\ell)_p\right) \simeq 0$ for most of the pixels. In Figure 3.8 we plot the histograms of the entropy values for the masks before and after the refinement step so as to inspect any improvement. The two plots in the top row of Figure 3.8 show the histograms for the masks of Figure 3.2b, while the histograms shown in the bottom row correspond to the masks after the refinement step. Clearly, the refinement step "cleans" the masks significantly.

3.7 Discussion

As we have seen, training the generative model for multiple objects using a direct search over all $J_b J_f^L$ values of the transformation variables is not feasible. Rather than use approximate simultaneous inference of the hidden variables we have developed a sequential method which extracts the background and foreground objects one-at-a-time from the input images. This is achieved by robustifying the generative model so that occlusions of either foreground or background can be tolerated. The results above show that this greedy algorithm is very effective at finding the background and foreground objects in the data.

Furthermore, although we have described this work in relation to image modelling, it can be applied to other domains. For example, one can apply the greedy approach to fitting mixture models, as we will describe in Chapter 5. Also, one can make a model for sequence data by having Hidden Markov models (HMMs) for a "foreground" pattern and the "background". Faced with sequences containing multiple foreground patterns, one could extract these patterns sequentially using a similar algorithm to that described above. It is true that for sequence data it would be possible to train a compound HMM consisting of $L+1$ HMM components simultaneously, but there may be severe local minima problems in the search space so that the sequential approach might be preferable.

A criticism about the above framework for learning multiple objects in images concerns the size of the transformation space i.e. the values of J_f and J_b. Even when

we learn a single object at a time the possible transformations of the object that we need to consider can be enormous. For example, considering only translations in units of one pixel gives P transformations. If the object can be rotated, we might need to consider about 360 rotations which immediately gives $360P$ total transformations. This means that for 100×100 images we will need $P = 3.6 \times 10^6$ transformations. For richer spaces of transformations, such as the affine transformations, we will need to consider a much larger number of transformations and the method will be extremely slow. A way of tackling this problem is to decompose the whole transformation into simpler transformations and then use a variational approximation to separate the search. Frey and Jojic (2001) used this approach to search over translations, rotations and scalings, combined with Fast Fourier Transforms to learn a single object. A more advanced technique based on this idea is that of Winn and Blake (2005) who use full affine transformations by globally searching over translations, rotations and scalings and then considering an additional local search over an affine transformation. However, this technique has so far been demonstrated only for a single foreground object against a static background.

When the images come from a video sequence with multiple moving objects we can speed up the search over the transformations based on the fact that the motion of an object between successive frames is constrained to be small. Considering video data and using the above constraint, in the next chapter we present a method that greatly speeds up the greedy algorithm for learning multiple objects and allow us to consider full affine transformations.

Chapter 4

Fast learning of multiple objects and parts from video

The greedy algorithm for learning multiple objects presented in the previous chapter works on unordered sets of images. In this case training can be very slow as it is necessary to search over all possible transformations of at least a single object on every image. However, for video sequences we could considerably speed up the training by first tracking the objects before knowing their full structure. Tracking can approximate the underlying sequence of transformations of an object in the frames and thus learning can be carried out with a very focused search over the neighbourhood of these transformations or without search at all when the approximation is accurate.

In this chapter we develop an algorithm that works in conjunction with the greedy algorithm so that the stage where the greedy algorithm learns an object (by assuming unordered images) is now speeded up by applying first tracking and then learning given the approximated transformations. First, the moving background is tracked and learned, and moving foreground objects are found at later stages. The tracking algorithm itself recursively updates an appearance model of the tracked object so that occlusion is taken into account, and approximates the transformations by matching this model to the frames through the sequence. This provides accurate transformations, e.g. in the experiments in section 4.5, we learn the objects without search using only the transformations found by the tracking algorithm and obtain good results.

Furthermore, in this chapter we learn articulated parts of the human body from video data using unsupervised learning. We apply the same algorithm for learning multiple objects in order to learn articulated parts, so that the parts are first tracked and then have their full structure learned. We also learn a distribution over parts' transformations, that expresses all the valid relative movements of the parts. In section 4.5 we assume that parts can undergo full affine transformations and we extract three parts (the two arms and the head/torso) from a video sequence of the upper human body. Much of the work of this chapter has been described in Titsias and Williams (2004).

The structure of the remainder of the chapter is as follows: In section 4.1 we briefly review work done using tracking for learning multiple moving objects in video sequences. In section 4.2 we present the training algorithm based on tracking, while in section 4.3 we describe learning of parts. In section 4.4 we discuss related work regarding both tracking multiple objects and learning parts of objects. In section 4.5 we give experimental results, and we conclude with a discussion in section 4.6.

4.1 Tracking multiple moving objects

Consider a video sequence where each frame contains the view of a single object against clutter (multiple objects will be discussed shortly). The constraint that the object does not move very fast, hence it cannot change its instantiation parameters rapidly, can be encoded using a Hidden Markov Model (HMM) where the hidden states correspond to the transformations of the object and the observations are the video frames (Jojic et al., 2000). The number of transformations I_f used to explain the motion in a frame of the video can be much smaller compared to the global number of transformations as now we only need to consider a window of neighbouring transformations centred at the transformation of the previous frame. EM can be applied for learning the object appearance as shown by Jojic et al. (2000). In the E-step the inference over transformations is carried out by running the forward-backward algorithm in $O(NI_f^2)$ time, where N is the number of frames. In the case of multiple moving objects the probabilistic generative model described in section 2.2.3 can be suitably modified to

generate the images similarly to the HMM. Particularly, the model now will be a factorial HMM type of model (Ghahramani and Jordan, 1997). For L moving objects, the inference step of the EM algorithm needs $O(N(I_f^2)^L)$ time, which grows exponentially with the number of objects similarly to the case of unordered (independently drawn) images.

However, even when we have a single moving object the number of transformations I_f can be of order of ten of thousands (e.g. if we discretize a full affine transformation over some window of neighbouring transformations). Thus, the quadratic term I_f^2 makes the computations prohibitive. For example, Jojic et al. (2000) have applied their HMM approach for learning a single object considering only translations. A solution to the above problem is to have a separate method for approximating the MAP estimate of the sequence of transformations of the HMM. Many methods achieve this considering a recursive processing of the video frames and applying motion estimation and tracking. In the following of the section we briefly discuss such techniques from the literature that have been applied for learning 2D appearance models of multiple objects in video sequences.

An algorithm for finding the transformations of objects in a video can be based on motion estimation in pairs of successive frames. We refer to these techniques as two-frame motion estimation methods. First, two-frame motion estimation is carried out through the whole sequence and then the object transformation at each frame is estimated as a cumulative sum of all the motions up to that frame[1]. This basically is the approach considered by several methods that learn multiple moving objects represented as 2D layers (appearances and support masks) in image sequences (Wang and Adelson, 1994; Darrell and Pentland, 1995; Sawhney and Ayer, 1996). These methods use optical flow estimation (Horn and Schunck, 1981) between pairs of frames and introduce the concept of multiple moving objects by constraining the flow vectors to follow a parametric model such as a simple translational or affine motion (Bergen et al., 1992; Wang and Adelson, 1994; Black and Anandan, 1996). A segmentation of the first frame into coherently moving regions is also needed since in order to estimate the motion of an object we need to know the image area that belongs to that object.

[1]This implies that the first frame in the sequence is the reference frame. Alternatively, we could choose some other frame as the reference frame and compute the transformations accordingly.

Motion estimates and segmentation are updated iteratively until the first frame is segmented into stable moving regions and the corresponding motions are approximated. k-means has been used for the segmentation step (Wang and Adelson, 1994) as well as Gaussian mixture models (Jepson and Black, 1993; Sawhney and Ayer, 1996; Weiss and Adelson, 1996; Vasconcelos and Lippman, 2001).

The main drawback of all two-frame motion estimation methods is that they strongly depend on the accuracy of the motion estimates when matching successive frames. Partial occlusion of the objects affects the accuracy of the motion estimation. Particularly, occlusion that occurs in both the first and the second frame can have an additive effect in the accuracy, e.g. when some part of an object is visible in some frame, while only the remaining part is visible in the next frame, the two-frame approach will not be able to estimate the motion. In addition, the widely-used optical flow algorithm suffers from the aperture problem. Since the transformation of an object at each frame is finally approximated as the cumulative sum of the motions up to that frame, small errors in the motion estimates will accumulate, resulting in a poor approximation of the transformations.

Motion estimation can be improved by accumulating an appearance model of a moving object through time (Irani et al., 1994; Tao et al., 2000). In this case a stabilized appearance model of the object is stored and updated as we process the frames and the motion of each frame is found by matching the stored appearance model to the frame. We refer to these methods as model-based tracking methods. Notice that in this case we can directly compute the transformation of an object by matching the stored appearance to the current frame without needing to take an accumulated sum of motions up to that frame. To whole approach can be much more reliable than the two-frame tracking framework. For example, consider the situation were we compute wrongly the transformation of an object at some frame due to severe occlusion. In other words we lose the track of the object at that frame. The two-frame method will provide wrong estimates of the transformations after that frame. On the other hand the model-based approach can still recover the track as long as in some subsequent time we manage to correctly match the object model to the current frame. Next we present a novel model-based type algorithm for tracking multiple objects.

4.2 Speeding up the greedy algorithm using tracking

In this section we present a tracking algorithm that applies to a sequence of frames and approximates the corresponding set of transformations of the objects. The method is combined with the greedy algorithm so that we track and learn all the differently moving objects sequentially. We start by tracking the background, while the foreground objects are ignored (using robust statistics) at this stage. Once the transformation of the background in all frames has been approximated we learn its full structure through a focused search. Note that this procedure simply replaces the step 1 of the greedy algorithm as summarized in section 3.3.4. Given that we know the background enables us to track and learn foreground objects, so that step 3(a) of the greedy algorithm is modified suitably.

Section 4.2.1 discusses how we can track the background while section 4.2.2 describes tracking of the foreground objects.

4.2.1 Tracking the background

We wish first to track the background and ignore all the other motions related to the foreground objects. To introduce the idea of our tracking algorithm assume that we know the appearance of the background $\mathbf{b}$ as well as the transformation j_b^1 that associates $\mathbf{b}$ with the first frame. Since motion between successive frames is expected to be relatively small we can determine the transformation j_b^2 for the second frame by searching over a small discrete set of neighbouring transformations centred at j_b^1 and inferring the most probable one (i.e. the one giving the highest likelihood given by equation (3.5), assuming a uniform prior). This procedure can be applied recursively to determine the sequence of transformations in the entire video.

However, the background $\mathbf{b}$ is not known in advance, but we can still apply roughly the same tracking algorithm by suitably initializing and updating the background $\mathbf{b}$ as we process the frames. More specifically, we initialize $\mathbf{b}$ so that the centred part of it will be the first frame $\mathbf{x}^1$ in the sequence. The remaining values of $\mathbf{b}$ take zero values and are considered as yet not-initialized which is indicated by a mask $\mathbf{m}$ of the same size as $\mathbf{b}$ that takes value 1 for initialized pixels and 0 otherwise. The transformation of

the first frame j_b^1 is the identity, which means that the first frame is untransformed. The transformation of the second frame and in general any frame $t+1$, $t \geq 1$, is determined by evaluating the posterior

$$R(j_b^{t+1}) \propto \exp\left\{ \sum_{p=1}^{P} (T_{j_b^{t+1}} \mathbf{m}^t)_p \log p_b(x_p^{t+1}; (T_{j_b^{t+1}} \mathbf{b}^t)_p) \right\}, \tag{4.1}$$

over the set of possible j_b^{t+1} values around the neighbourhood of j_b^t. Note that (4.1) is similar to the likelihood (3.5) with the only difference that pixels of the background that are not initialized yet are removed from consideration. Once we know j_b^{t+1}, we use all the frames up to the frame $\mathbf{x}^{t+1}$ to update $\mathbf{b}$ according to

$$\mathbf{b}^{t+1} \leftarrow \sum_{n=1}^{t+1} [T_{j_b^n}^T (\mathbf{r}_{j_b^n}^n * \mathbf{x}^n)] ./ \sum_{n=1}^{t+1} [T_{j_b^n}^T \mathbf{r}_{j_b^n}^n], \tag{4.2}$$

where the $\mathbf{r}_{j_b^n}^n$ vectors have been updated according to equation (B.3) (see appendix B) for the old value $\mathbf{b}^t$ of the background. Notice that the above update is given by equation (B.4) in the greedy algorithm, by considering only the first $t+1$ frames as the training data and assuming that the found transformation values $\{j_b^1, \ldots, j_b^{t+1}\}$ take all the probability ($P(j_b^i|\mathbf{x}^i) = 1$, $i = 1, \ldots, t+1$). The mask $\mathbf{m}$ is also updated so that it always indicates the pixels of $\mathbf{b}$ that are explored so far.

As we process the frames the background $\mathbf{b}$ is adjusted so that any occluding foreground object is blurred out revealing the background behind. An illustrative example of applying the algorithm is shown in Figure 4.2a. Having tracked the background, we can then learn its full structure assuming a focused search over the neighbourhood of the approximated transformations. What we mean by learning using a focused search is that we apply an EM type of algorithm for learning the background as that mentioned in section 3.3.1 and further presented in appendix B.1. But now we suitably constrain each posterior probability $P(j_b|\mathbf{x}^n)$ to obtain non-zero values only in a neighbourhood of transformations centred at j_b^n that is found by the tracking algorithm. Appendix C formally presents this algorithm with focused search as a variational EM.

4.2.2 Tracking the foreground objects

Assume that the background has now been learned. The pixels which are explained by the background in each image $\mathbf{x}^t$ are flagged by the background responsibilities $\mathbf{r}_{j_b^t}^t$

(computed by equation (B.3)). Clearly the mask $\overline{\mathbf{r}}^t_{j_b^t} = \mathbf{1} - \mathbf{r}^t_{j_b^t}$ roughly indicates all the pixels of frame $\mathbf{x}^t$ that belong to the foreground objects. By focusing only on these pixels we wish to start tracking one of the foreground objects through the entire video sequence and ignore for the moment the rest foreground objects.

Our algorithm tracks the first object by simultaneously updating its mask $\boldsymbol{\pi}_1$ and appearance $\mathbf{f}_1$. The mask and the appearance are initialized so that $\boldsymbol{\pi}_1 = \mathbf{0.5} * \overline{\mathbf{r}}^t_{j_b^t}$ and $\mathbf{f}_1 = \mathbf{x}^1$, where $\mathbf{0.5}$ denotes the vector with 0.5 values[2]. Due to this initialization we know that the first frame is untransformed, i.e. j_1^1 is the identity transformation. To determine the transformation of the second frame and in general the transformation j_1^{t+1}, with $t \geq 1$, of the frame $\mathbf{x}^{t+1}$ we find the most probable value of j_1^{t+1} according to the posterior

$$R(j_1^{t+1}) \propto \exp\left\{ \sum_{p=1}^{P} (\mathbf{w}_1^{t+1})_p \log\Big((T_{j_1^{t+1}} \boldsymbol{\pi}_1)_p \times p_f(x_p^{t+1}; (T_{j_1^{t+1}} \mathbf{f}_1)_p) + (\mathbf{1} - T_{j_1^{t+1}} \boldsymbol{\pi}_1)_p (1 - \alpha_b) U(x_p^{t+1}) \Big) \right\}, \quad (4.3)$$

where $\mathbf{w}_1^{t+1} = \overline{\mathbf{r}}^{t+1}_{j_b^{t+1}}$. $R(j_1^{t+1})$ measures the goodness of the match at those pixels of frame $\mathbf{x}^{t+1}$ which are not explained by the background. Note that as the objects will, in general, be of different sizes, the probability $R(j_1^{t+1})$ over the transformation variable will have greater mass on transformations relating to the largest object. Recall that $p_f(x_p^{t+1}; (T_{j_1^{t+1}} \mathbf{f}_1)_p)$ includes an outlier component so that some badly misfit pixels can be tolerated.

Once we determine j_1^{t+1} we update both the mask $\boldsymbol{\pi}_1$ and appearance $\mathbf{f}_1$. These updates are similar to those given by (B.9) and (B.10), with the only difference that we use only the first $t+1$ frames and the approximate transformations to obtain posterior probabilities $P(j_1^t | \mathbf{x}^t) = 1$. Note that the updates are robust to occlusion (because of the semantics of $\mathbf{r}^n_{j_1^n}$; see appendix B.2) so that occlusion of the tracked object can be tolerated. This makes tracking reliable even for the video frames the object is occluded. Note also that as the frames are processed tracking becomes more stable since $\boldsymbol{\pi}_1$ approximates the mask of a single object and the $\mathbf{f}_1$ will contain a sharp and clear view

[2]The value of 0.5 is chosen to express our uncertainty about whether these pixels will ultimately be in the foreground mask or not.

for only the one object being tracked while the rest of the objects will be blurred; see Figure 4.2b for an illustrative example.

Once this first object model has been learned we can go through the images to find which pixels are explained by this model, and update the $\mathbf{z}$ vector accordingly as explained in section 3.3 (see also appendix B). We can now run the same tracking algorithm again by updating $\mathbf{w}^t_{\ell+1}$ ($\ell \geq 1$) as by $\mathbf{w}^t_{\ell+1} = \mathbf{z}^t_\ell * \mathbf{w}^t_\ell$ which allows tracking a different object on the $\ell+1$th iteration. Note also that the new mask $\boldsymbol{\pi}_{\ell+1}$ is initialized to $\mathbf{0.5} * \mathbf{w}^t_{\ell+1}$ while the appearance $\mathbf{f}_{\ell+1}$ is always initialized to the first frame $\mathbf{x}^1$.

4.3 Learning about parts

For the recognition of complex objects that contain several parts that can take on different configurations relative to each other, it has long been recognized that a strategy based on the recognition of the individual parts and their relationships is likely to be advantageous; see e.g. Biederman (1987) on recognition-by-components. Much work in computer vision along this track uses manually identified parts; recent examples of such work are Felzenszwalb and Huttenlocher (2005) and Schneiderman and Kanade (2004) which consider people and cars, and faces and cars respectively. In contrast we are interested in learning parts from data. Much less work has been done on this topic, but some specific contributions are discussed in section 4.4.2.

Section 4.3.1 discusses how the greedy algorithm is used for learning parts of articulated objects using video data. Section 4.3.2 describes a method for computing a joint distribution over parts that can undergo full affine transformations.

4.3.1 Learning parts using the greedy algorithm

To learn objects parts using unsupervised learning from some training images we should first ask the question what properties a set of pixels should have in order to be considered as a single part. An general answer should be that pixels having strong statistical correlation[3] observed through the whole data set, should be taken as a single

[3] Statistical correlation between some quantities can be quantified in terms of the mutual information which measures our certainty about predicting one quantity by observing others.

part. Of course, turning this general principle into a specific algorithm to learn parts from images can be a great challenge. Here we focus on articulated objects (e.g. a human body) where the statistical correlation is due to co-motion so that a set of pixels explained across the data set by the same transformation (such as a combination of translation and rotation) should be taken as a single part.

The greedy algorithm combined with tracking can be used to discover the appearances and masks of the parts of an articulated object from video data. Figure 4.1 shows three frames of the arms-torso video data in which we applied the greedy algorithm and Figure 4.4 displays the three parts that are found. However, notice that the greedy algorithm was first developed to train a generative model for multiple objects which considers the objects as being apriori independent. Particularly, as explained in section 3.1, the location (decided by the transformation) of each object in the image is assumed to be generated independently from the locations of the other objects. This assumption is a good approximation for multiple objects which their motion is largely unrelated. However, for object parts this assumption is inappropriate since the parts much be located within the image so that to form a valid object. For example, in the data of Figure 4.1 the arms should always be connected to the torso in a way that a valid human body appearance is expressed. In the next section we discuss how the information provided by the greedy algorithm can be used to compute a joint distribution over the transformations of the parts.

4.3.2 Finding the joint distribution of the parts

Assume now that we have applied the greedy algorithm and we have discovered L parts of a single object. Particularly, for each part ℓ the algorithm outputs the appearance $\mathbf{f}_\ell$, the mask $\boldsymbol{\pi}_\ell$ and the transformations $(j_\ell^1, \ldots, j_\ell^N)$ that describe how the parts are instantiated in each training image. Using this information we wish to learn the joint distribution $P(j_1, \ldots, j_L)$ over the transformations of the parts.

The relationship between the parts is independent from the location of the whole object within the image. For example, for the arms-torso data of Figure 4.1 the parts are connected through flexible joints according to some rules that are invariant from the global location of the human in the image. Next we exploit this property by computing

Figure 4.1: Three frames of the arms-torso video sequence.

a distribution over parts in some canonical frame and considering an additional global transformation to instantiate the whole object in any location within an image.

When each part can only translate, the location (instantiation within the image) of each object part can be described by depicting one landmark on the part appearance and translating this landmark. Now when the parts undergo full affine transformations we need to depict three non co-linear landmarks on each part appearance and then consider transformed positions of these landmarks. Figure 4.6a illustrates the original (untransformed) coordinates of the landmarks overlaid on each part appearance for the arms-torso data, while Figure 4.6b (bottom row) shows the corresponding transformed coordinates of the landmarks when several upper human body images are generated.

Let $\mathbf{t}_\ell^n = (\mathbf{t}_{\ell,1}^n, \mathbf{t}_{\ell,2}^n, \mathbf{t}_{\ell,3}^n) = [(\mathbf{t}_{\ell,1}^n)_x \ (\mathbf{t}_{\ell,1}^n)_y \ (\mathbf{t}_{\ell,2}^n)_x \ (\mathbf{t}_{\ell,2}^n)_y \ (\mathbf{t}_{\ell,3}^n)_x \ (\mathbf{t}_{\ell,3}^n)_y]^T$ be the coordinates of all three landmarks of the ℓth part in the image $\mathbf{x}^n$. Using the data set $(\mathbf{t}_\ell^1, \ldots, \mathbf{t}_\ell^N)$, where $\ell = 1, \ldots, L$, of the landmarks in all training images we wish to compute the canonical frame distribution. This distribution is defined in some reference frame so that one of the parts will be always untransformed. Without loss of generality, we choose the first part as the reference part and thus the distribution is defined over all landmarks of the parts (transformed into the canonical frame space) except the landmarks of the reference part. Particularly, we need to remove the effect of the transformation of the reference part according to $\mathbf{o}_{\ell,i}^n = (A_1^n)^{-1}\left(\mathbf{t}_{\ell,i}^n - \mathbf{m}_1^n\right)$, with $i = 1,2,3$ and $\ell = 2, \ldots, L$, and where the 2×2 matrix A_1^n and $\mathbf{m}_1^n = [(\mathbf{m}_1^n)_x \ (\mathbf{m}_1^n)_y]^T$ denote the affine parameters for the reference part in the image $\mathbf{x}^n$. Thus, the canonical frame distribution is computed using the data set $(\tilde{\mathbf{o}}^1, \ldots, \tilde{\mathbf{o}}^N)$ where each $\tilde{\mathbf{o}}^n = (\mathbf{o}_2^n, \ldots, \mathbf{o}_L^n)$.

We model this distribution by the following mixture of Gaussians

$$Q(\tilde{\mathbf{o}}) = \sum_{k=1}^{K} P_k N(\tilde{\mathbf{o}}; \boldsymbol{\mu}_k, W_k W_k^T + v_k I), \tag{4.4}$$

where each $\boldsymbol{\mu}_k$ is a $6(L-1)$ mean vector and the $6(L-1) \times 6(L-1)$ covariance matrix $W_k W_k^T + v_k I$ is parameterised by the low rank $6(L-1) \times q$ matrix W_k (where $q < 6(L-1)$) and the variance v_k. Each Gaussian in equation (4.4) is a probabilistic PCA model (Tipping and Bishop, 1999; Roweis, 1998), thus the whole mixture can represent a non-linear data manifold as a collection of local linear models. Notice that due to the articulation motion of the parts, we expect each vector of landmarks (i.e. $\tilde{\mathbf{o}}$) to lie on a non-linear manifold. In Figure 4.5 we illustrate this fact by showing several two-dimensional plots of different pairs of dimensions of the data set used to estimate the distribution (4.4) for the arms-torso sequence (see also section 4.5.2). Clearly, the plots in Figure 4.5 suggest that the manifold in this case has an non-linear arc shape. To specify the parameters $(\boldsymbol{\mu}_1, W_1, v_1, \ldots, \boldsymbol{\mu}_K, W_K, v_K)$, we use as a training set the landmarks $(\tilde{\mathbf{o}}^1, \ldots, \tilde{\mathbf{o}}^N)$ and maximize the log likelihood using the EM algorithm (Tipping and Bishop, 1999).

To instantiate an object within an image, we first draw a sample $\tilde{\mathbf{o}}$ from the canonical frame distribution Q and then we choose an affine transformation for the reference part and transform all the landmarks. Particularly, selecting the affine $(A_1, \mathbf{m}_1)$ for the reference part by a discrete uniform distribution P_{ref}, each landmark in an observed image is given by $\mathbf{t}_{\ell,i} = A_1 \mathbf{o}_{\ell,i} + \mathbf{m}_1$. Thus, the distribution in the image space will be

$$p(\mathbf{t}_1, \tilde{\mathbf{t}}) = P_{ref}(\mathbf{t}_1) \sum_{k=1}^{K} P_k N(\tilde{\mathbf{t}}; \tilde{A}\boldsymbol{\mu}_k + \tilde{\mathbf{m}}, \tilde{A}\Sigma_k(\tilde{A})^T), \tag{4.5}$$

where $\tilde{\mathbf{t}} = (\mathbf{t}_2, \ldots, \mathbf{t}_L)$, $\tilde{\mathbf{m}}$ is a $6(L-1)$-dimensional vector constructed by $3(L-1)$ replicates of the translation $\mathbf{m}_1$, $\tilde{A}$ is a $6(L-1) \times 6(L-1)$ bock-diagonal matrix with the blocks being $3(L-1)$ replicates of the A_1 matrix and $\Sigma_k = W_k W_k^T + v_k I$. Note that equation (4.5) is the desired distribution over transformations of the parts i.e. $P(j_1, \ldots, j_L)$.

The above framework can be used in the case the parts undergo more restricted (or more general) transformations than a full affine, e.g. if we consider only translations, rotations and scalings, then the only difference in computing the canonical frame distribution is that we need only two landmarks on each part instead of three. So far we

have not discussed how we choose the original (untransformed) locations of the landmarks on the part appearances. We have an automatic method for doing this which is explained in the experiments described in section 4.5. For example, this method has been used to choose the landmarks above the head/torso and the two arms shown in Figure 4.6a.

4.4 Related work

In this section we discuss related work. In particular, section 4.4.1 discusses related work regarding tracking multiple objects and section 4.4.2 about learning parts.

4.4.1 Tracking

There is a huge literature on motion analysis and tracking in computer vision, and there is indeed much relevant prior work. Particularly, Wang and Adelson (1994) estimate object motions in successive frames and track them through the sequence by computing optical flow vectors, fit affine motion models to these vectors, and then cluster the motion parameters into a number of objects using k-means. Darrell and Pentland (1995); Sawhney and Ayer (1996) uses similar approaches based on optical flow estimation between successive frames and apply also the MDL principle for selecting the number of objects. Note that a major limitation of optical-flow based methods concerns regions of low texture where flow information can be sparse, and when there is large inter-frame motion. The method of Irani et al. (1994) is much more relevant to ours. They do motion estimation using optical flow by matching the current frame against an accumulative appearance image of the tracked object. The appearance of a tracked object develops though time, although they do not take into account issues of occlusion, so that if a tracked object becomes occluded for some frames, it may be lost.

The work of Tao et al. (2000) is also relevant in that it deals with a background model and object models defined in terms of masks and appearances. However, note that the mask is assumed to be of elliptical shape (parameterised as a Gaussian) rather than a general mask. The mask and appearance models are dynamically updated. How-

ever, the initialization of each model is handled by a "separate module", and is not obtained automatically. For the aerial surveillance example given in the paper initialization of the objects can be obtained by simple background subtraction, but that is not sufficient for the examples we consider. Later work by Jepson et al. (2002) uses a *polybone* model for the mask instead of the Gaussian, but this still has limited representational capacity in comparison to our general mask. Jepson et al. also use more complex tracking methods which include the birth and death of polybones in time, as well as temporal tracking proposals.

The idea of focusing search when carrying our transformation-invariant clustering has also been used before, e.g. by Fitzgibbon and Zisserman (2002) in their work on automatic cast listing of movies. However, in that case prior knowledge that faces were being searched for meant that a face detector could be run on the images to produce candidate locations, while this is not possible in our case as we do not know what objects we are looking for apriori.

As well as methods based on masks and appearances, there are also feature-based methods for tracking objects in image sequences, see e.g. Torr (1998), Wills et al. (2003). These attempt to track features through a sequence and cluster these tracks using different motion models. We believe that these methods are of considerable interest, particularly when combined with new ideas on features that are stable to various transformations such as scaling and rotation (Lowe, 2004). We will further discuss this in section 6.3 where we provide directions for future work.

4.4.2 Parts

In section 4.4.2.1 we discuss related work about learning objects parts and in section 4.4.2.2 about modelling the spatial relationships of the parts.

4.4.2.1 Unsupervised learning of parts

Perona and co-workers have developed important work on unsupervised learning of objects and parts through a series of papers (Burl et al., 1998; Weber et al., 2000; Fergus et al., 2003). Their general approach (as described in Weber et al., 2000) is as follows: First an interest operator is used to locate keypoints in an training set of

images each showing an object view against clutter then graylevel descriptors of these keypoints are extracted. These features are then clustered to give a number of "part" types; note that there can be several detections of a given part type in a given image. The learning of this model is slow as one has to deal with the combinatorics of the assignment of all part type detections in an image to the part types in the model. Note that in the end the model only prescribes the feature appearance at certain locations in the image, and does not provide a full appearance model of the object. However, an important aspect of this work is that object classes (e.g. cars) can be learned, not just specific objects.

Shams and von der Malsburg (1999) have also developed a method for learning parts by matching images in a pairwise fashion, trying to identify corresponding regions in the two images. These candidate image patches were then clustered to compensate for the effect of occlusions. We make the following observations on their work: (i) in their method the background must be removed otherwise it would give rise to large match regions; (ii) their data (although based on realistic CAD-type models) is synthetic, and designed to focus learning on shape related features by eliminating complicating factors such as background, surface markings etc.

Lee and Seung (1999) described a non-negative matrix factorization method to tackle part decompositions. However, this interesting work does not deal with the problem of parts undergoing transformations, and so could not extract the kinds of parts found by our method. Cooperative Vector Quantization methods (Hinton and Zemel, 1994; Ghahramani, 1995; Ross and Zemel, 2003) use probabilistic models to find parts decompositions of data, but again they do not consider parts undergoing transformations.

4.4.2.2 Joint model for the parts

Burl et al. (1998) considered a Gaussian distribution for connecting the parts that can only deal with a translational relative deformation of the parts[4]. Particularly, they only consider a single landmark point on each part, so that only translations of the parts are

[4]Note that the work of Weber et al. (2000) and Fergus et al. (2003) uses the same framework for connecting the parts with this introduced by Burl et al. (1998).

modelled. Thus, this method is not suitable for parts that undergo different rotations (i.e. having more than one articulation points) such as parts of a human body. Felzenszwalb and Huttenlocher (2005) presented a method for modelling the joint distribution over parts that can vary due to translations rotations and scalings. In particular, they consider a tree-structured Markov Random Field over transformation parameters of the parts with each potential function being a product of three Gaussians (two for the landmark that explains translation and one for the scale parameter) and a Von Mises distribution that models rotation angle. However, this method can only be trained with labelled examples e.g. they need explicitly a human to label the position of the joint of each pair of connecting parts in each training image. Note that our method models full affine relative deformation of the parts by depicting three landmarks on each part and additionally the selection of the landmarks is done automatically.

There are many methods in the literature that make use of a set of landmarks to express a statistical model of an object's shape, e.g see Grenander et al. (1991), Cootes et al. (1992), Williams (1994), Bregler and Omohundro (1994) and Heap and Hogg (1995) among others. These methods depict landmarks usually along the boundary of the object and then model shape deformation by expressing a joint distribution over these landmarks. For example, Cootes et al. (1992) assume a linear model by estimating the covariance matrix from the data using the first few principal components. Bregler and Omohundro (1994) used clustering combined with PCA to deal with the non-linear data manifold, similarly to the probabilistic PCA mixture model we used in section 4.3.2. Heap and Hogg (1995) transform suitably chosen landmarks into to polar coordinates so as to make the data manifold linear in the case of rotational deformation. Note that in our case the landmarks are used to express relative relationship or deformation between parts rather than deformation of the object as a whole. This means that the number of landmarks scales with the number of parts and it can be much smaller than the number of landmarks used in all the above methods. Since we model deformation of articulated (largely rigid) parts, the method of Heap and Hogg (1995) may be an alternative way to model the canonical frame distribution.

4.5 Experiments

In section 4.5.1 we demonstrate the greedy and tracking algorithm for learning multiple objects in video and we compare it with the original greedy algorithm (see chapter 3) with respect to running time complexity. In section 4.5.2 we demonstrate the method for learning the joint distribution over the transformations of the parts.

4.5.1 Demonstration of the tracking algorithm

We will consider two video sequences: the Frey-Jojic (FJ) sequence available from `http://www.psi.toronto.edu/layers.html` and the arms-torso video sequence showing a moving human upper body (see Figure 4.1).

In terms of computational time, the greedy algorithm without tracking requires $O(JNM)$ operations per learning an object, where J is the number of transformations, N the number of frames and M the number of EM iterations needed for convergence. Here one operation consists of computing $p_b(x_p; b_p)$ or $p_f(x_p; f_p)$ (according to equations (3.2) and (3.1)) for all image pixels. The algorithm using tracking needs roughly $O(I_1 N + N^2 + I_2 N M_2)$ operations per pixel, where I_1 is the number of transformations considered for searching to find the transformation of the next frame as we track the object through the sequence, I_2 is the number of transformations of the focused search in the learning stage, and M_2 is the corresponding number of EM iterations. The N^2 term is due to updates of the mask and appearance during tracking, since at each time we consider all the frames processed so far. Notice that $J_f \gg I_1$, I_2 so the tracking algorithm should enjoy considerable speedups.

The FJ sequence consists of 44 118×248 images (excluding the black border); it was also used at the previous chapter (see Figure 3.2a). The results in Figure 4.3 were obtained using a 15×15 window of translations in units of one pixel during the tracking stage ($I_1 = 225$) and a 1×1 window ($I_2 = 1$), i.e. without search, around the approximated location during the learning stage. This learning stage requires EM which converged in about $M_2 = 30$ iterations. Figure 4.2a shows the evolution of the initial appearance of the background ($t = 1$) through frames 10 and 20 as we track the background. Note that as we process the frames the background becomes sharp

and clear, while the foreground objects are blurred. Similarly, Figure 4.2b shows the evolution of the mask and appearance as we track the first object (Frey). Again notice that as we process the frames the mask focuses on only one of the two objects and the appearance remains sharp only for this object.

The algorithm with tracking needs a total of $13,156$ operations to find an object. The original greedy algorithm considers all 118×248 possible translations i.e. $J_f = 29264$, and also requires $M_1 = 70$ iterations to reach convergence. Thus, the total number of operations required by the original greedy algorithm is $90,133,120$, which implies that we gain a speed up of over 6850 for learning a single object. The real running times of our MATLAB implementations roughly confirm the above, since the greedy algorithm learns the whole sequence in 80 hours, while the algorithm using tracking runs in 3 minutes.

We demonstrate our method for learning parts of human body using the arms-torso sequence that consists of 79 76×151 images. Three frames of this sequence are shown in Figure 4.1. To learn the articulated parts we use full affine transformations so that the transformation matrix $T_{j\ell}$ that applies to $\boldsymbol{\pi}_\ell$ and $\mathbf{f}_\ell$ implements an affine transformation. We implemented this using the MATLAB function *tformarray.m* and considering nearest neighbour interpolation; see also appendix A.1. Note that to discretize an affine transformation we use the fact that any affine is a combination of a rotation, two scalings (one for each image axis), another rotation and a translation[5]. However, for this particular sequence similar results can be obtained by constraining the affine transformation to be only a combination of a translation and a rotation, since the parts actually only translate and rotate.

When we use only translations and rotations the tracking method uses a 10×10 window of translations and 10 rotations (at 4^o spacing) so that it searches over $I_1 = 1000$ transformations in total. After tracking a part we use a focused search around a $1 \times 1 \times 1$ window (i.e. no search) to learn its full structure. Figures 4.4 shows the three parts discovered by the algorithm i.e. the head/torso and the two arms. Note that the ambiguity of the masks and appearances around the joints of the two arms with

[5]This is based on the fact that the 2×2 A matrix that linearly transforms a pixel location, can be decomposed using Singular Value Decomposition as $A = U\Lambda V$, where U is the left rotation matrix, Λ a diagonal scaling matrix and V the right rotation matrix.

$t = 1$ $t = 10$ $t = 20$

(a)

$t = 1$ $t = 10$ $t = 20$

(b)

Figure 4.2: Panel (a) shows the evolution of the background appearance $\mathbf{b}$ during tracking at time 1 (initial value of $\mathbf{b}$), 10, and 20. Clearly, as we track the background the appearance $\mathbf{b}$ reveals the background behind the foreground objects which are blurred. Similarly, panel (b) shows the evolution of the mask $\boldsymbol{\pi}_1$ (top row) and the appearance $\mathbf{f}_1$ (bottom row) at times 1, 10 and 20 as we track the first objects (Frey). Again notice how the mask becomes focused on one of the objects (Frey) and how the appearance remains clear and sharp only for Frey.

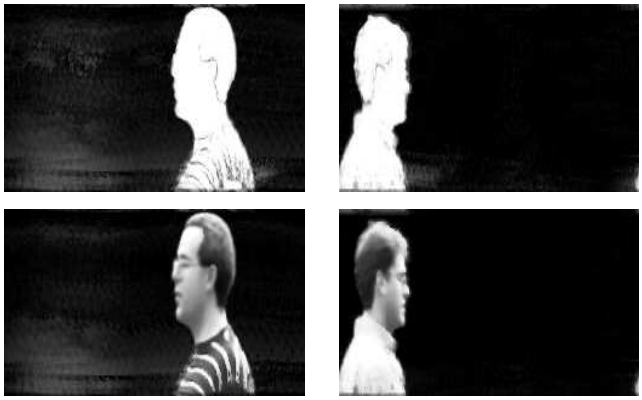


Figure 4.3: The final masks and the element-wise product of the mask and appearance model ($\boldsymbol{\pi} * \mathbf{f}$) learned for Frey (first column from the left) and Jojic (second column).

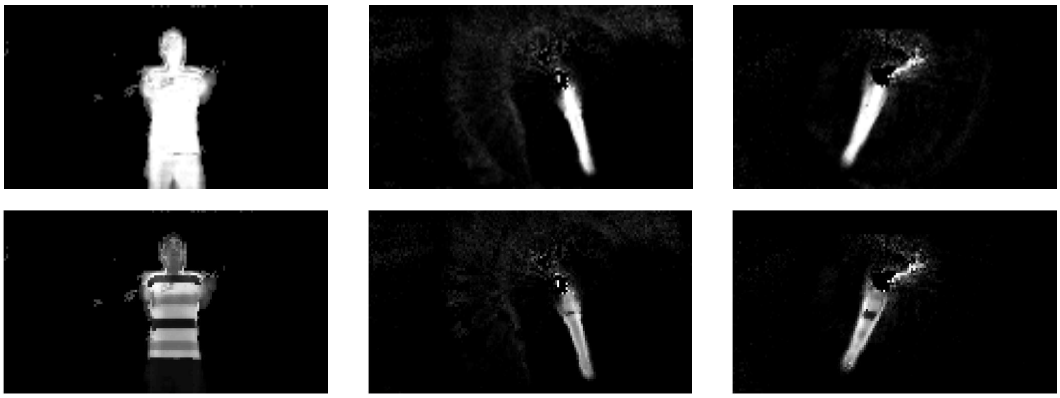


Figure 4.4: The masks and appearances of the parts of the arms-torso video sequence. The plots in the first column show the learn mask (top row) and the element-wise product of the mask and appearance (bottom row) for the head/torso. Any pair of panels in the other two columns provides the same information for the two arms.

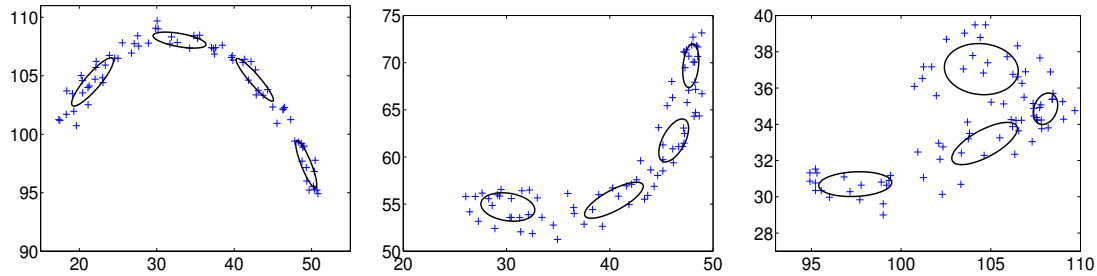
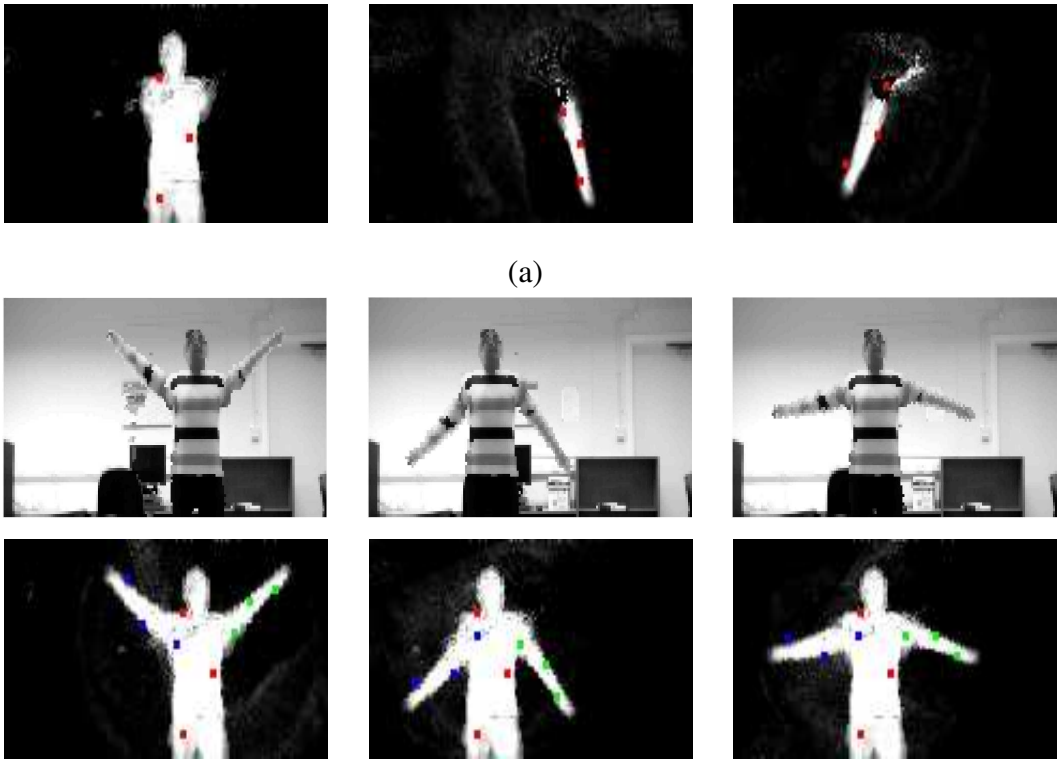


Figure 4.5: Three scatter plots of different pairs of dimensions of the 12-dimensional data set used to estimate the canonical frame distribution for the arms-torso sequence. The left panel plots the two dimensions of the landmark that belongs to the first arm (showing in the middle column of Figure 4.4) and it is the closest to the hand. Similarly, the middle panel plots the dimensions of the landmark of the second arm that is again the closest to the hand. The right panel plots two dimensions that belong to landmarks of different arms. One standard deviation Gaussian ellipses of the probabilistic PCA mixture model are also plotted (by picking the corresponding two dimensions of the means and the 2×2 parts of the covariance matrices).

the torso which is due to the deformability of the clothing in these areas. The total real running time for learning this sequence was roughly half hour. When we use full affine transformations the algorithm becomes slower and runs roughly in 10 hours. Notice also that running the original greedy algorithm on this sequence is infeasible in practice. In particular, the greedy algorithm without tracking requires consideration of all possible 76×151 translations combined with, say, 200 rotations which immediately gives a very large number of transformations ($J_f = 2,295,200$) that should be considered.

4.5.2 Demonstration of learning the joint distribution over parts

The tracking algorithm for learning the objects parts outputs a set of appearances and masks as well as the approximated transformations for all the training images. In order now to apply the framework for computing a joint model for the parts, as described in section 4.3.2, we need first to depict the landmarks on each part appearance. We do this automatically using the learned masks. Particularly, from the mask $\boldsymbol{\pi}_\ell$ we consider all



(a)

(b)

Figure 4.6: Panel (a) shows the landmarks depicted on each part mask for the arms-torso video data according to the method described in the text. Panel (b) illustrates three samples of human upper body appearances. Particularly, the top row shows the appearances produced by the joint model for the parts and the trained generative model, while the bottom row shows the corresponding positions of the transformed landmarks (using different colours for different parts) superimposed on the transformed masks.

the pixel locations that belong to the object part (the mask gives a value close to 1 for those pixels) and we choose the landmarks so that to form a triangle that covers a large amount of the part area. Particularly, by applying PCA we select the first landmark to be across the minor axis and close to the boundary of the object part (indicated by the corresponding eigenvalue). For example, this is how the first from the right landmark is depicted on the head/torso mask as shown in Figure 4.6a. The other two landmarks are selected to be across an axis that is parallel to the principal axis and also close to the boundary side that is in the opposite direction than the side of the first landmark; see again Figure 4.6a for an illustration. Note that the landmarks can be outside of the object, and in fact many alternative choices of the landmarks can be equal good.

Having selected the landmarks as described above, the joint distribution over parts is computed as discussed in section 4.3.2. Particularly, as explained in section 4.3.2, we form a data set of vectors each one corresponding to each training image and containing the landmarks of all the parts except the landmarks of the reference part and transform these vectors into the canonical frame space by applying the inverse transformation of the reference part. For example, for the arms-torso video sequence the head/torso[6] was selected as the reference part and thus the above data set consists of six landmarks corresponding to the two arms, which gives vectors into the 12-dimensional space. Figure 4.5 shows three scatter plots of this 12-dimensional data set by picking pairs of different combinations of the dimensions. Clearly, Figure 4.5 indicates that the landmarks of the two arms in the canonical frame space form an arc shaped manifold due the rotational motion.

The canonical frame distribution described by equation (4.4) was chosen to have 4 components where for each component a 4-dimensional PCA space was used. These choices were based on a validation set. To optimize the parameters of this distribution we applied EM (Tipping and Bishop, 1999) and particularly we used the NETLAB implementation which is available from `http://www.ncrg.aston.ac.uk/netlab`. Figure 4.5 illustrates how the mixture components fit the data. To inspect the quality of the fit to the underlying shape distribution, we draw several samples of human upper body appearances based on the learned joint distribution over the parts. From visual in-

[6] We chose the reference part so as to have the largest area computed using the mask.

spection these samples always look plausible shapes, e.g. Figure 4.6b shows three such samples. Note that to produce each plot in the top row in Figure 4.6b we first sample the landmarks according to the equation (4.5), which provides all the transformations, and then an object appearance is generated according to the probability model of equation (2.15). In the bottom row of Figure 4.6b we also show the corresponding positions of the landmarks superimposed on the masks. Note that the arms are always placed symmetrically to each other since that was also the nature of the training examples.

4.6 Discussion

Above we have shown how to use a tracking method to greatly speed up the greedy algorithm presented in chapter 3 and we also demonstrated that this method can identify articulated parts of objects, as in the human body. Additionally, we described how to connect the parts by learning a prior distribution over the transformations of the parts.

Our current tracking algorithm for learning multiple objects has certain limitations. First of all, the objects we learn are specific objects without significant internal variability and viewpoint changes. For example, consider the arms-torso data of Figure 4.1, the parts largely undergo 2D motion and no 3D rotations and substantially changes of the viewpoint of the object are considered. In case the viewpoint changes so that the object appearance changes dramatically, the probabilistic model as well as the tracking algorithm will need some modification. An additional remaining issue is to automatically specify the number of objects L that exists in the sequence and also to detect when a model is a part or an independent object. Suggestions for future research on all the above topics such as specifying the number of objects, learning objects/parts with internal variability and finding non-articulated parts will be discussed at the conclusions chapter and specifically in section 6.3.

Chapter 5

Greedy training of mixtures models using robust statistics

The key idea of the greedy algorithm for learning multiple objects presented in chapter 3 is that it uses robust statistics in order to learn one object at each time. As explained in section 3.3, the greedy algorithm trains sequentially a certain type of a mixture model for image pixels where each component corresponds to an object model, so that at each stage a component fits a subset of the pixels and the remaining pixels are explained by an outlier process.

In this chapter we discuss how we can fit a mixture model used for clustering in a similar manner. Particularly, we incorporate an outlier component (an uniform density to any possible data point) into the mixture model which allows us to fit one cluster of the data by "ignoring" the rest of the clusters. Data points that are fitted by a model are then "removed from consideration" in a probabilistic fashion so that at the next stage a new model can fit to an unexplored region of the data space and so on. Such a method can be useful for improving parameter initialization since it provides a sensible way to sequentially initialize each component model to data regions that are not well explained by the already fitted models.

An additional feature of the algorithm is that it can indicate when to stop adding new components; when the outlier component fits no data (or fits only background clutter data) we have potentially reached the desirable number of components and we

can stop. We show that this can be used to find the number of components in some simple clustering problems.

The structure of the remainder of the chapter is as follows: In section 5.1 we describe the sequential algorithm for fitting a mixture model assuming any form for the component density models. In section 5.2 we discuss related work, while in section 5.3 we show some experiments in real data sets using Gaussian and multinomial mixtures and provide also a comparison with the regular EM. For Gaussian mixtures we include also in the comparison another sequential (greedy) algorithm proposed by Vlassis and Likas (2002) and Verbeek et al. (2003). We conclude with a discussion in section 5.4.

5.1 Sequential algorithm for mixture models

We address the problem of learning a mixture density model with J components

$$P(\mathbf{x}) = \sum_{j=1}^{J} \pi_j P_j(\mathbf{x}|\theta_j), \tag{5.1}$$

where $P_j(\mathbf{x}|\theta_j)$ is the jth component having parameters θ_j and π_j the mixing coefficient. Mixture models have been widely used in statistical modelling as density estimation methods McLachlan and Peel (2000). Given a set of i.i.d data $X = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ we wish to estimate the underlying density of $\mathbf{x}$ by a mixture model of the form (5.1). The component densities $P_j(\mathbf{x}|\theta_j)$ can be chosen from some parametric family such as the exponential family. Most of the presentation in the rest of the paper assumes that $P_j(\mathbf{x}|\theta_j)$ can be any distribution while in out experiments we specify this to be a multivariate Gaussian in the case of continuous data and a multinomial for discrete-valued data.

In this section we describe the sequential algorithm for learning mixture models. Particularly, section 5.1.1 illustrates the idea of using an outlier component to train a single Gaussian density. Section 5.1.2 describes the algorithm for sequentially fitting the components of a mixture model using an outlier component and section 5.1.3 provides details regarding the application of the algorithm to Gaussian and multinomial mixtures.

5.1.1 Fitting one density model together with an outlier component

We wish to learn a density model $P(\mathbf{x}|\theta)$ together with a uniform distribution $U(\mathbf{x})$, called the outlier component, so that

$$P(\mathbf{x}) = \alpha P(\mathbf{x}|\theta) + (1-\alpha)U(\mathbf{x}). \tag{5.2}$$

For clarity assume at the moment that the model $P(\mathbf{x}|\theta)$ is a multivariate Gaussian with parameters $\theta = \{\boldsymbol{\mu}, \Sigma\}$. Selecting first a value for α we can learn the parameters by maximizing the log likelihood $L = \sum_{n=1}^{N} \log P(\mathbf{x}^n)$ using the EM algorithm.

Notice that if $\alpha = 1$ the outlier component is neglected and the parameter estimate for the Gaussian is the maximum likelihood solution. As α decreases the Gaussian becomes more and more focused on some population of the data and as α approaches zero the Gaussian fits very few data points ending up with fitting just one data point.

An obvious use of this model is to robustify the Gaussian estimate by choosing a high value for α (say 0.9) which can be useful in situations where one data cluster is embedded in background clutter. However, what is less obvious is the fact that by choosing properly the value of α the robust model of equation (5.2) can be used for learning just one data cluster by ignoring any other clusters of the data. To illustrate this consider the data of Figure 5.1 which form three separate clusters. We maximize the likelihood by initializing $\boldsymbol{\mu}$ selecting one data point and $\Sigma = cI$ with c equal to the maximum variance of all data dimensions. α is initialized to 0.5 and is learned infrequently by EM (every 10 iterations). Figure 5.1 illustrates two runs of the EM algorithm under two different parameter initializations of the mean.

If we can fit just one cluster of the data by the model described above we can then, by repeating the process, fit all the data clusters sequentially. This motivates the sequential algorithm for fitting mixture models described in next section.

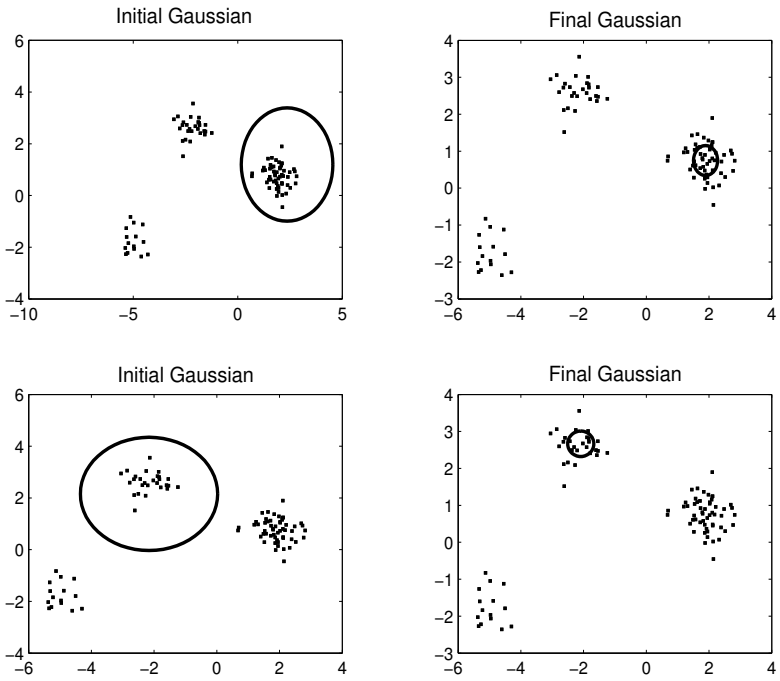


Figure 5.1: Illustrates fitting a Gaussian using an outlier component. Under different initializations of the mean (plots on the left) we can discover different clusters (plots on the right).

5.1.2 Fitting mixture models sequentially

In this section we discuss how we can use an outlier component to fit a mixture model rather than a single density model. Such a mixture should have the form

$$P(\mathbf{x}) = \sum_{j=1}^{J} \pi_j P_j(\mathbf{x}|\theta_j) + (1 - \sum_{j=1}^{J} \pi_j) U(\mathbf{x}), \tag{5.3}$$

where generally $\sum_{j=1}^{J} \pi_j \leq 1$. We wish to train this mixture model sequentially by learning only one density model $P_j(\mathbf{x}|\theta_j)$ at each stage. An intuitive way of thinking about this is that we start by assuming that the mixing coefficients π_j, $j,\ldots,J$ are set to zero, so that the outlier component has all the probability. At jth stage the mixing coefficient π_j is set free to get a positive value and the corresponding component model $P_j(\mathbf{x}|\theta_j)$ is allowed to fit some part of the data. At each stage the mixing coefficient of the outlier component always decreases which implies that the probability of what is considered as outlying data decreases sequentially.

Attias (2000) has shown how mixture models can be trained using variational Bayesian methods. It is interesting to note that if one component of the mixture at a time is updated repeatedly while keeping the other components fixed then a scheme quite similar to ours would emerge, as the as-yet-unfitted components would tend to have vague distributions not dissimilar to our uniform component[1].

We now describe our algorithm in detail, starting with training the first component $P_1(\mathbf{x}|\theta_1)$. By allowing the coefficient π_1 to obtain a positive value we have the mixture

$$P(\mathbf{x}) = \pi_1 P_1(\mathbf{x}|\theta_1) + (1 - \pi_1) U(\mathbf{x}), \tag{5.4}$$

which is exactly the model discussed in section 5.1.1 and thus learning the parameters $\{\theta_1, \pi_1\}$ can be done by maximizing the log likelihood using EM.

Suppose now that we have fitted a model $P_1(\mathbf{x}|\theta_1)$ to the data. What we wish to do next is to train the second mixture component $P_2(\mathbf{x}|\theta_2)$ by considering the mixture

$$P(\mathbf{x}) = \pi_1 P_1(\mathbf{x}|\theta_1) + \pi_2 P_2(\mathbf{x}|\theta_2) + (1 - \pi_1 - \pi_2) U(\mathbf{x}). \tag{5.5}$$

This case now becomes a little more complicated in the sense that we wish the second model not to fit data that are already well explained by the first model. Generally the

[1] We thank Steve Roberts for a helpful discussion on this point.

new model $P_2(\mathbf{x}|\theta_2)$ should fit a subset of the data that is reasonably different from all the data fitted by $P_1(\mathbf{x}|\theta_1)$. Such a constraint can be efficiently taken into account by applying a constrained EM algorithm where instead of the log likelihood we maximize a lower bound of the log likelihood (Neal and Hinton, 1998). Particularly, we compute the responsibilities of the uniform component for each data point $\mathbf{x}^n$ by

$$z_1^n = \frac{(1-\pi_1)U(\mathbf{x}^n)}{\pi_1 P_1(\mathbf{x}^n|\theta_1) + (1-\pi_1)U(\mathbf{x}^n)}, \tag{5.6}$$

which are computed by having only trained the first component and then we express a lower bound of the log likelihood of the model (5.5). Particularly, introducing the notation $P_2^u(\mathbf{x}) = \pi_2 P_2(\mathbf{x}|\theta_2) + (1-\pi_1-\pi_2)U(\mathbf{x})$ and using Jensen's inequality we have

$$\begin{aligned} & \sum_{n=1}^{N} \log\{\pi_1 P_1(\mathbf{x}|\theta_1) + P_2^u(\mathbf{x})\} \\ = & \sum_{n=1}^{N} \log\left\{(1-z_1^n)\frac{\pi_1 P_1(\mathbf{x}|\theta_1)}{1-z_1^n} + z_1^n \frac{P_2^u(\mathbf{x})}{z_1^n}\right\} \\ \geq & \sum_{n=1}^{N}(1-z_1^n)\log \pi_1 P_1(\mathbf{x}^n|\theta_1) + \sum_{n=1}^{N} z_1^n \log P_2^u(\mathbf{x}) \\ + & H(\{z_1^n\}), \end{aligned} \tag{5.7}$$

where $H(\{z_1^n\})$ denotes an entropic term independent of $\{\pi_2, \theta_2\}$. Since the parameters of the first model are fixed, maximizing the above bound simplifies to maximizing only the second term in the above sum under the constraint that π_2 should receive a value smaller or equal to $1-\pi_1$. According to the form of the above objective function maximizing with respect to $\{\theta_2, \pi_2\}$ favours solutions where the model fits data that previously was explained mainly by the outlier component. To make this more obvious note that the weights $\{z_1^n\}$ are close to zero for all data explained by $P_1(\mathbf{x}|\theta_1)$ and close to one for all data explained by the outlier component. So the objective function (5.7) effectively removes from consideration in a probabilistic fashion data fitted by the first model. This process of fitting the mixture components to the data can be performed sequentially. The algorithm is summarized below

1. Set $j = 0$. Initialize: $z_0^n = 1$ for all n.

2. Set $j = j+1$. Initialize θ_j and $\pi_j = \alpha(1 - \sum_{i=1}^{j-1} \pi_i)$, where $\alpha < 1$ (we use $\alpha = 0.5$).

3. Optimize the parameters $\{\theta_j, \pi_j\}$ by running EM and maximizing:

$$\sum_{n=1}^{N} z_{j-1}^n \log\{\pi_j P_j(\mathbf{x}^n|\theta_j) + (1 - \sum_{i=1}^{j-1} \pi_i - \pi_j) U(\mathbf{x}^n)\}, \tag{5.8}$$

   where π_j is sparsely updated by EM (every 10 iterations).

4. Update the log likelihood weights

$$z_j^n = \frac{(1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x}^n)}{\sum_{i=1}^{j} \pi_i P_i(\mathbf{x}^n|\theta_i) + (1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x}^n)}. \tag{5.9}$$

5. Go to step 2 or output the mixture $P_j(\mathbf{x}) = \sum_{i=1}^{j} \pi_i P_i(\mathbf{x}|\theta_i) + (1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x})$.

At each stage of the above algorithm a new model is trained (step 3) by maximizing a weighted log likelihood where the weights $\{z_j^n\}$ mask out (probabilistically) data fitted by the previously learned models. At step 4 the $\{z_j^n\}$ values are updated so that at the next stage we can fit a new model most probably to a different data subset. The sequential process can be considered as a kind of boosting algorithm for density estimation as the data points are reweighted on each iteration so that points that are well fitted by the current models become less important in later stages.

Note that the mixing coefficient of the outlier component $1 - \sum_{i=1}^{j} \pi_i$ can only decrease at each stage and naturally the learning process stops once this coefficient becomes very close to zero. In section 5.3.2 we describe a simple stopping criterion based on this idea and we use it to find the number of clusters in some simple clustering problems.

The outcome of the sequential algorithm can be used to initialize a mixture model. In such case the outlier component is discarded and the coefficients π_j, $j = 1, \ldots, J$ are normalized to sum to one. The parameters of the resulting mixture can be refined by applying EM and maximizing the likelihood.

5.1.2.1 Refinement of the previous learned components

So far the models fitted to the data remain fixed for the later stages, however it might be more effective if we can refine their parameters during learning. Next we discuss the refinement procedure we use in our implementation.

The refinement can be introduced as an additional step of the sequential algorithm between steps 4 and 5 which is applied only for $j \geq 2$. After step 4 the values $\{(1-z_j^n)\}$ are the responsibilities according to which all the components $P(\mathbf{x}|\theta_i)$, $i = 1,\ldots,j$ explain the data. Based on these responsibilities we maximize a lower bound of the log likelihood

$$\begin{aligned} F &= \sum_{n=1}^{N}(1-z_j^n)\log\sum_{i=1}^{j}\pi_i P(\mathbf{x}^n|\theta_i) \\ &+ \sum_{n=1}^{N} z_j^n \log(1-\sum_{i=1}^{j}\pi_i)U(\mathbf{x}^n) + H(\{z_j^n\}) \\ &= \sum_{n=1}^{N}(1-z_j^n)\log\sum_{i=1}^{j}\pi_i P(\mathbf{x}^n|\theta_i) + const, \end{aligned} \tag{5.10}$$

where the sum $\sum_{i=1}^{j}\pi_i$ is fixed at the value that obtains before we apply the refinement step. Note that the second term in the first line is a constant since $\sum_{i=1}^{j}\pi_i$ is invariant as well as the entropic term since depends on the $\{z_j^n\}$ values. Thus, the refinement objective function is just the first term in the above sum under the constraint that $\sum_{i=1}^{j}\pi_i$ is invariant, which can be carried out using EM. Such refinement will be within the data regions that the first j components already fit. Completely unexplored data regions for which the outlier component obtains high responsibility will remain unexplored after refinement.

Assuming that we have carried out the refinement step as described above, we need to feed the changes made back into the sequential algorithm. This is simply done by updating the weights, so the refinement step is completed by updating $\{z_j^n\}$ according to the step 4 of the sequential algorithm.

It is interesting to compare the weighted log likelihood used in the refinement step (equation (5.10)) with the corresponding used for learning a new Gaussian (equation (5.8)). The quantity F in (5.10) explicitly masks out (in a probabilistic way) all the data fitted by the outlier component. This allows refinement of the Gaussians without

big moves in unexplored data regions. On the other hand the weighted log likelihood used to learn a new Gaussian works in the opposite way; it masks out all the data fitted by the already learned Gaussians in order to fit a new Gaussian to unexplored data regions.

5.1.2.2 Running time analysis

For mixture models trained by EM algorithm the time complexity depends on the number of evaluations of the component densities $P_j(\mathbf{x})$. The sequential algorithm without refinement fits one component at each stage which has complexity $O(N)$ per EM iteration. If the maximum number of iterations is M_1, the whole process for fitting J models needs $O(JNM_1)$ operations. This can be further reduced by noting that at each stage the data fitted by the current models become much less important since their weights z_{j-1}^n will take a very close to zero value. We can explicitly remove these data from the dataset by expressing the set $A_j = \{n : z_{j-1}^n > \delta\}$ with δ small (e.g. $\delta = 0.05$) and use only the data from A_j when we train the jth component. Thus, the jth stage needs $O(N_j M_1)$ time ($N_j = |A_j|$) and the total time complexity becomes $O(JKM_1)$ where K denotes the average value of all N_js. In practice K can be much smaller than N. The algorithm with refinement steps at the jth stage learns first one component and updates the current mixture (only for $j \geq 2$) which totally requires $O(JNM_1 + \sum_{i=2}^{J} iNM_2) = O(JNM_1 + J^2NM_2)$ operations where M_2 is the maximum number of EM iterations for the refinement procedure. This time can also be reduced similarly to the no-refinement case.

Note that when we train a J-component mixture the sequential algorithm is used as the initialization procedure of the mixture model. This means that it is not desirable to run all the EM algorithms required by the sequential algorithm till convergence. Typically, we can apply few iterations M_1 and M_2 for training a new Gaussian and refining the current mixture, respectively. Especially, it is desirable that M_2 be small (e.g less than 20) so that the refinement procedure to be fast. Once the components have been initialized the regular EM is applied to refine this mixture which needs $O(JN)$ operations per EM iteration.

5.1.3 Parameter initialization and specifying $U(\mathbf{x})$

When we apply the sequential algorithm we have to specify several parameters. First of all the α value used in the step 2 of the algorithm is set equal to 0.5. Now, when $P_j(\mathbf{x}|\theta_j)$ is a Gaussian we initialize the mean to some training data point selected from the distribution $G(n) = z_j^n / \sum_{n'=1}^{N} z_j^{n'}$. $G(n)$ at each stage favours points that are mainly explained by the outlier component. The covariance matrix is initialized to be spherical with variance equal to the maximum variance of all data dimensions.

In the case of multinomial mixtures the component densities have the form

$$P_j(\mathbf{x}|\theta_j) = \prod_{i=1}^{d} P(x_i|\theta_j^i), \tag{5.11}$$

where d is the number of dimensions and $P(x_i|\theta_j^i)$ are multinomial distributions. To initialize θ_j we first select a training data point $\mathbf{x}^n$ from $G(n)$ and initialize the multinomial parameters for dimension i by giving a fraction γ of the probability mass to the value that corresponds to the value of x_i^n and split the rest of probability mass uniformly over the rest of the values.

One crucial point is how we define the uniform distribution $U(\mathbf{x})$. An obvious way is to find the hypercube/sphere that contains all the training data and express the uniform density in that space. However, in high dimensional spaces the data often lies on lower dimensional manifolds and such a "bounding box" uniform distribution will tend to give very low probability densities to the data points in comparison with the other components. To overcome this, we set $U(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} P(\mathbf{x}^n|\theta_{ML})$ where $P(\mathbf{x}|\theta_{ML})$ is a single component model (e.g. a Gaussian) using maximum likelihood parameters θ_{ML}. This can be seen as a rescaling of the α parameter in equation (5.2).

5.2 Related work

In addition to the standard EM algorithm, various initialization strategies have also been proposed. Cheeseman et al. (1993) and Figueiredo and Jain (2002) among others have demonstrated a backwards selection method, starting with many components and pruning some away using Bayesian type of model selection criteria. The forward

sequential (greedy) algorithm for Gaussian mixtures proposed by Vlassis and Likas (2002) and Verbeek et al. (2003) initializes the Gaussians one after the other by comparing at each stage a set of candidate initializations. Once a Gaussian has been initialized the current mixture model is refined by maximizing the likelihood using EM. One important difference with our method is that we use the outlier component which allows the Gaussians at each stage to fit some part of the data, while in Vlassis and Likas (2002); Verbeek et al. (2003) the Gaussians at each stage fit all the data. Note also that our method does not use a set of candidate initializations when we fit a new component and for this reason it requires less running time.

The sequential algorithm can be regarded as a boosting density estimation algorithm. There has been some recent work (Thollard et al., 2002; Meek et al., 2002; Rosset and Segal, 2003) on extending boosting from the supervised learning problem to the density estimation problem. Our sequential formulation of fitting process is reminiscent of these boosting algorithms. However, one attractive feature of our scheme is that the boosting view derives from a constrained EM formulation of the problem which at each stage derives the weights of the data points based on the outlier component.

5.3 Experiments

In this section we demonstrate the sequential algorithm for training Gaussian and multinomial mixtures. In section 5.3.1 we apply the algorithm to learn a J-component mixture on two real datasets and provide a comparison with the regular EM as well as the sequential algorithm of Verbeek et al. (2003). In section 5.3.2 we apply the sequential algorithm to find the number of components in a case with well-separated clusters.

5.3.1 Training a J-component mixture model

We present two experiments in real high dimensional data using Gaussian and multinomial mixtures respectively. In the first experiment we use the Brodatz textures images following an experimental setup used in Verbeek et al. (2003). The task is to cluster

a set of $16 \times 16 = 256$ patches taken from 256×256 pixel Brodatz texture images. We consider the number of clusters (textures) from which patches are extracted to be $J = \{3,5,7,9\}$. For a specific J we randomly choose J textures from the 37 available textures, create a set of $900J$ patches and then keep the half $(450J)$ for training and the rest for testing. We repeat this experiment 50 times. Each data set was also projected from the 256 to 50 dimensions using PCA in order to speed up the experiment. For each of the 50 datasets of a certain J we train a mixture model with J components using (i) k-means initialized EM[2] ($k$means), (ii) the sequential (greedy) algorithm of Verbeek et al. (2003) (VVK)[3], (iii) the sequential algorithm with refinement (Ref) and with (iv) no refinement (NoRef) steps. For the Ref and NoRef methods the M_1 and M_2 numbers defined in section 5.1.2.2 are set to 50 and 20 respectively. Table 5.1 and 5.2 display the t-statistic values of the difference of the average log likelihoods for training and test data set respectively. Note that when we consider the differences in log likelihoods of the method A and B ($A - B$ in the notation in the Tables 5.1 and 5.2) and the t-statistic is larger than 2.01 the method A is significantly better than B at level 5% ($t_{0.025,49} = 2.01$). When the t-statistic is less than -2.01 the method A is significantly worse. Observe that the sequential fitting algorithm with refinement is better than EM initiated with k-means and that these differences are significant for $J = 7$, 9. Also the method with refinements is better than the VVK method and this is significant for $J = 7,9$. Note that using refinement always improves the results compared to the algorithm with no refinement steps. We have also run the algorithm for mixture fitting proposed by Figueiredo and Jain (2002) on this data using their code (available from `http://www.lx.it.pt/~mtf/mixturecode.zip`). However, the pruning strategy they use means that one cannot guarantee to get J components in the final model, and when fewer than J components are selected the log likelihoods are low leading to poor performance in comparison to the methods reported in the Tables.

In our experiments the regular EM with few steps of the k-means algorithm for initialization was the fastest followed by the sequential algorithm with no-refinement and the algorithm with refinement, while the method of Verbeek et al. (2003) was the

[2] We used the NETLAB implementation available from `http://www.ncrg.aston.ac.uk/netlab`.

[3] The code is provided by the authors at http://carol.science.uva.nl/∼vlassis/research/learning/ index_en.html.

	3	5	7	9
NOREF - *k*MEANS	-1.03	-0.27	1.7	1.74
REF - *k*MEANS	0.74	1.74	**3.26**	**3.5**
VVK - *k*MEANS	1.05	-0.26	1.27	0.48
REF - NOREF	**2.34**	1.24	**2.63**	**3.48**
REF - VVK	0.12	1.38	**2.49**	**2.49**
NOREF - VVK	-1.46	-0.07	0.83	1.04

Table 5.1: t-statistic values for the differences of the average training set log likelihoods for the Brodatz textures. Bold face indicates that the test is significant at the 5% level.

	3	5	7	9
NOREF - *k*MEANS	-1.02	-0.23	1.66	1.52
REF - *k*MEANS	0.69	1.75	**3.17**	**3.25**
VVK - *k*MEANS	1.01	-0.24	1.03	0.18
REF - NOREF	**2.34**	1.3	**2.59**	**3.8**
REF - VVK	0.11	1.41	**2.57**	**2.53**
NOREF - VVK	-1.43	-0.05	0.93	1.09

Table 5.2: t-statistic values for the differences of the average test set log likelihoods for the Brodatz textures. Bold face indicates that the test is significant at the 5% level.

slowest. For example, choosing $J = 7$ the real running time of the algorithms (averaged over 10 runs) was: 25 seconds for the *k*means method, ii) 30 seconds for the NoRef method, iii) 65 seconds for the Ref method and iv) 102 seconds for the VVK method, respectively.

The second experiment uses handwritten digits data as employed in Frey et al. (1996). The digits are quantized to 8×8 binary images. Following Meila and Heckerman (2001) we use only the digit 6 dataset (but note that different preprocessing means that our results are not directly comparable to theirs). This dataset consists of 700 training cases and 200 test cases. The implementation of the regular EM (Reg-EM) is based on initializing the multinomial parameters by picking J data points randomly and applying the parameter initialization method described in section 5.1.3 (γ was cho-

J	REF	NOREF	REG-EM
3	-28.1 ± 0.21	-27.97 ± 0.2	-28.06 ± 0.21
5	-26.8 ± 0.03	-26.79 ± 0.03	-26.8 ± 0.03
7	-26.23 ± 0.09	-26.25 ± 0.13	-26.28 ± 0.14
9	-25.93 ± 0.14	-25.90 ± 0.12	-25.95 ± 0.14
11	-25.72 ± 0.11	-25.68 ± 0.09	-25.67 ± 0.12

Table 5.3: Mean average log likelihoods for the test data in the digit6 dataset.

sen to be 0.75). The mean average log likelihoods over 20 random initializations and for different choices of the number of components J are displayed in Table 5.3. On this data there is no significant difference between the performance of the algorithms.

5.3.2 Finding the number of components

The sequential algorithm can indicate when to stop adding components, since in case the outlier component fits very few data we probably have reached the required number of components. We apply this to find the number of components in cases of separate clusters.

The criterion we use is very simple. Assume that we have trained the jth Gaussian, we express the set $S_j = \{\mathbf{x}^n : z_j^n > 0.5\}$ which indicates the data points for which the outlier component obtains the largest responsibility. Now if S_j is a large set, we know that there are some data regions that are not explored by the Gaussian models, so we have to continue learning. However, when this set contains very few data we can conclude that the Gaussians have discovered all the data regions. In our experiments we assume that 2% of the data is background noise, thus we require $|S_j| > 0.02N$, where N is the number of training data, otherwise we stop learning. For the data of Figure 5.2 generated from a five-component mixture we applied the sequential algorithm with refinement steps. The stopping criterion is met when we reach five components, when, in fact $|S_5| = 0$.

Note that the above criterion can find the number of components in problems with well-separated clusters. In cases the data form highly overlapping clusters or where

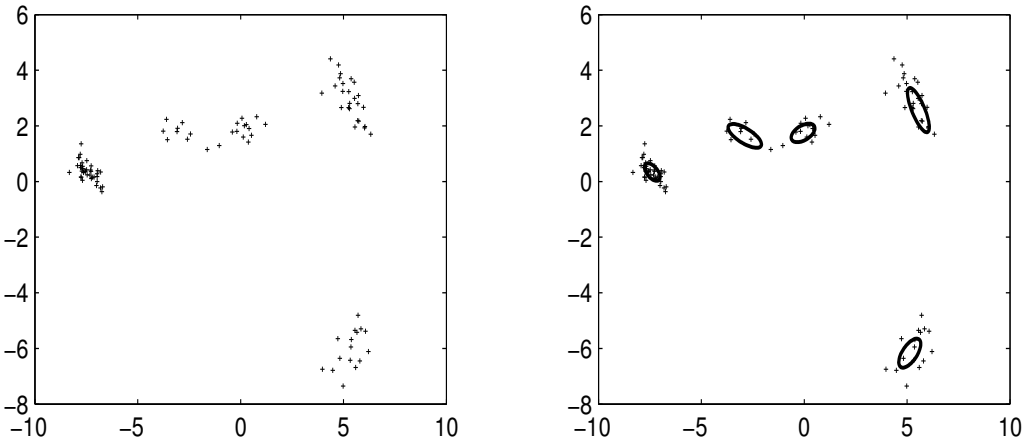


Figure 5.2: The plot on the left illustrates a data set generated from a 5-component Gaussian mixture. The plot on the right displays the solution found by the sequential algorithm once the stopping criterion is met.

there is no clear separation of the data into clusters the above criterion will only roughly indicate how many components needed to represent the data.

5.4 Discussion

Above we have described a sequential approach for fitting mixture models, and have demonstrated that for training a J-component mixture model it can produce better performance than rival algorithms. Compared to the sequential method of Verbeek et al. (2003) our method gave better performance and it is also fastest since we do not use a set of candidate initializations when we fit a new component. Note that our method could probably be significantly improved by having a small set of candidate initializations.

In terms of model selection, Bayesian methods using the marginal likelihood as a selection criterion or approximations such as BIC penalties are most common. While our method is unlikely to be able to compete with sophisticated Bayesian methods such as reversible jump MCMC Green (1995) on densities whose components are not well

separated, it does provide a much more rapid answer.

Chapter 6

General discussion

In section 6.1 we give a summary of the contributions of this thesis. In section 6.2 we draw some conclusions regarding the usefulness of using robust statistics for learning models sequentially. Finally, in section 6.3 we describe directions for future work.

6.1 Summary

In this thesis we have dealt with the problem of learning specific multiple objects and object parts from a set of images using unsupervised learning. The main contributions of our work are:

- The generative model for learning multiple objects, where each image is synthesized by a background object and multiple foreground objects that are represented by 2D appearances and masks. Each image pixel is generated by a mixture distribution over all objects so that objects strictly combine by occlusion.

- The greedy algorithm for training the generative model, which by utilizing robust statistics can discover the background and the foreground objects one at each time.

- The tracking algorithm that greatly speeds up the original greedy algorithm in video data. This algorithm tracks an object (first the background, and then each

foreground object) by updating an appearance model of the object and matching this model to the frames through the sequence.

- The method for learning parts of articulated objects. This method first uses the greedy algorithm to find the appearances of the parts and then the joint distribution over parts is modelled by a probabilistic PCA mixture model over some landmarks assigned above the parts so that full affine relative transformation of the parts is allowed.

- The greedy algorithm for fitting a mixture model for clustering using an outlier component, which is based on similar principles as the greedy algorithm for learning objects.

6.2 When is a greedy algorithm using robust statistics useful?

The use of robust statistics plays a central role in both the greedy algorithm for learning multiple objects (chapters 3 and 4) and the algorithm for fitting a mixture model (chapter 5). Notice that the greedy algorithm for learning multiple objects achieves an approximate training of an originally intractable factorial learning model. An important question concerns when learning the components of a model one-at-a-time is more beneficial than training the whole model simultaneously.

Learning a model (for multiple objects or clustering) sequentially using robust statistics is generally easier from an optimization point of view. This is because fewer parameters are fitted each time and the corresponding likelihood surface is a smoothed version of the original log likelihood[1] with all component models present. For this reason, local maxima problems are reduced since both the objective function surface is smoother and the search space is smaller. This point was confirmed by our experiments in all data sets where we applied the greedy algorithm for learning multiple objects. Particularly, by having a simple mechanism for parameter initialization for each object

[1] All the contributions of the component models (objects or component density parameters) that are not fitted yet are explained by an outlier process.

(see section 3.6) any run of the algorithm was successful without facing local maxima problems.

However, the above point is useful only if our algorithm is able to sufficiently approximate the desirable values of the model components, so that the refinement step (where all the components are simultaneously maximized using the values of the greedy algorithm as the initial estimate) can easily find these values. The desirable values are those that would be hypothetically obtained by a global maximization of the likelihood of the overall model. The greedy algorithm for learning multiple objects provides good estimates of the objects since each mask and appearance pair describes quite efficiently the view of a single object (see experiments in sections 3.6 and 4.5). The refinement step described in section 3.4 converges very quickly and improves the appearances of the objects (e.g. it cleans the masks as shown in section 3.6) which indicates that the greedy algorithm had already provided good object models. The greedy algorithm for clustering is less efficient with respect to the above point, e.g. the refinement step might not converge in a few iterations and the final mixture model might differ significantly from that provided by the greedy algorithm. However, this strongly depends on the characteristics of a data set, since a data set might not admit an obvious separation into (Gaussian type) clusters.

6.3 Directions for future work

In section 6.3.1 we discuss future improvements of the current framework for learning multiple objects. In section 6.3.2 we describe a learning scenario where our approach for learning multiple objects can be used as the first step for training an object categorization system from unsegmented images with multiple objects using unsupervised learning.

6.3.1 Further improvements on learning multiple objects

In our current algorithm for learning objects we assume that we know the number of foreground objects that exist in the images. Generally, we would like to learn this number automatically. One way to incorporate this into the greedy algorithm is to

introduce a model selection criterion as a score for the number objects and evaluate this criterion each time we learn an object. Bayesian criteria such as BIC and MDL (Schwarz, 1978; Rissanen, 1987) could be implemented.

One problem that can arise with our current approach is when the appearance of an object can change markedly due to changes in pose and self-occlusion. For example, in the beginning of a sequence we may see a front view of a car, while at the end we may see a side view. In this case the generative model as well as the tracking algorithm (see chapter 2, 3 and 4) should be modified. Particularly, a suitable enhancement of the generative model will be to introduce a set of mask and appearance pairs, each one associated with a different view (or aspect, Koenderink and van Doorn 1979) of an object. The tracking algorithm should be also modified to track an object even when the viewpoint changes. Once the transformations of an object are approximated we could learn the set of different views by clustering similarly to the approach for clustering images showing a single object in different poses (Frey and Jojic, 2003). Additionally, when for a fixed viewpoint the appearance of an object can still vary due to some internal deformation, e.g. a face that changes expressions, we need to further enhance our model to account for internal object variability. One way to achieve this is to assume that the appearance of each foreground object (i.e. $\mathbf{f}_\ell$) in each image is modelled by a low-dimensional eigenspace (Black and Jepson, 1996) or a factor analysis model (Jojic et al., 2003).

The tracking algorithm presented in chapter 4 operates in image pixels. Particularly, it approximates the transformations of the objects by updating masks and appearances and matching them to each frame. This can be slow especially for large images. An alternative and possibly much faster approach to approximate the transformations is to extract features from the original images and then perform the computations in this features space; see e.g. Torr (1998), Wills et al. (2003). An interesting approach toward this direction is to to use features that are stable to various transformations such as SIFT features (Lowe, 2004). Once such features are detected in all frames, then the objective will be to cluster them into different moving objects and thus compute the transformation of an object within an frame using the locations of the objects' detected features.

The approach for learning parts using the greedy algorithm assumes that we know when a detected model is an object part or an independent object. Thus, to fully automate our method we should be able to group different models together when they belong to the same object. A way to do this can be based on the mutual information. Particularly, since we expect parts of the same object to have significant statistical dependence we can compute the mutual information between pairs of models that have been discovered by the greedy algorithm and group them into the same object as long as their mutual information is sufficiently large.

6.3.2 Unsupervised object recognition from images with multiple objects

Consider the case where we have a set of video sequences with each sequence showing the views of multiple moving objects. Such data might come from very realistic environments (e.g. from cameras in public places). Moving objects, for example, can be humans and cars. Learning starts by extracting the multiple objects from each video sequence using the method presented at chapter 4. This procedure will output a set of specific objects and particularly an intensity appearance image of each object and a mask.

The next step will be to consider each learned appearance of an object as a data point of a new training set and cluster these data using a mixture model where each component generates the appearance as a collection of object parts. Specifically, assume that there is only one underlying object class (e.g. humans). We first introduce a probabilistic generative model (similar to that described in chapter 2) that considers the appearance of the object as a mosaic of non-overlapping parts. The variation of each part can be modelled by assuming a continuous latent variable (subspace variable) in some lower dimensional space similar to probabilistic PCA or factor analysis models and also some image transformation latent variable that makes the part invariant under few transformations (e.g. translations and scalings) around a certain location. Given that we know the number of parts we can fit this model to the data using a variational EM algorithm and discover parts of arbitrary sizes and shapes[2]. When there are mul-

[2]The object part that a given pixel is generated from is found automatically by EM.

tiple underlying object classes (e.g. humans and cars), then the probabilistic model becomes a mixture model of the above type which can be fitted by an EM kind of algorithm.

The above procedure will output a class-conditional model corresponding to each cluster that describes an underlying object category. However, so far such a class-conditional model assumes that the parts are independent. We can improve on this by learning a joint distribution that connects the parts. This can be done as follows. Once we have applied the clustering algorithm described in the previous paragraph, we use the objects falling within a specific cluster to learn a joint distribution for the configurations of the parts. For each object in the cluster and for each part we find the most probable configuration of the latent variables (the subspace variable and the transformation variable) and fit a distribution to these data. Specifically, by assuming that this distribution is factorized over the subspace variables and the transformation variables, we can fit a Gaussian to the subspace variables and also a mixture of Gaussians to the transformation variables (using landmarks as discussed in section 4.3.2). The estimated joint distribution will model the relationships of the parts, which together with the model trained at the clustering stage will provide the final class-conditional density model for the object class.

With the above framework we can train in an unsupervised way a generative model for object categories from images with multiple objects. In addition, we learn class-specific parts of arbitrary shapes from the data that are described by subspace manifolds and can also undergo transformations.

Appendix A

Transformation matrices and EM for one foreground object

A.1 Transformation matrices

In chapters 2, 3 and 4, when we described algorithms for learning objects, we store images as vectors and express image transformations through matrix multiplications. For example, the appearance of a foreground object $\mathbf{f}$ (see e.g. section 2.2.1) is a P-dimensional vector, where P is the size of the data image vector $\mathbf{x}$, and corresponds to a column-wise unfold of an $P_x \times P_y$ image. Furthermore, a transformation is represented by $T\mathbf{f}$, where T is a $P \times P$ matrix[1]. Denoting images by vectors and expressing transformations through matrix multiplications is useful to derive the EM algorithms needed for learning (see appendix A.2). However, we never need to naively express a transformation as a matrix multiplication or explicitly store a $P \times P$ transformation matrix. As explained below, the transformation matrices are sparse and thus sparse algebra can be used to efficiently carry out all the necessary computations.

We first discuss how we compute a transformation matrix from a set of transformation parameters that operate in the two-dimensional image coordinates space. A transformation can be thought as a two-dimensional warping or motion from a source (or

[1] In general, we could choose $\mathbf{f}$ to have different size than the data image $\mathbf{x}$, which will make the matrix T rectangular. For example, this will be case for a moving background $\mathbf{b}$, which is larger than $\mathbf{x}$; see section 2.2.2.

untransformed) image $\hat{I}(x,y)$ to the destination (or transformed) image $I(x,y)$, where (x,y) is a pixel location in the image coordinates and $\hat{I}$ and I can generally have different sizes. To compute this transformation, we should for each pixel location (x,y) at the destination image I ask where did the pixel come from the source image $\hat{I}$. This means that we need to find the location $(\hat{x},\hat{y})$ (that might not be an integer-valued pixel location) within the source image $\hat{I}$ and then specify the destination pixel value $I(x,y)$ using interpolation. For example, in case of an affine transformation, described by the parameters $(A,[t_x\ t_y]^T)$, where A is a two dimensional invertible matrix and $[t_x\ t_y]^T$ a translation, the destination pixel (x,y) and the source location $(\hat{x},\hat{y})$ are associated by

$$\begin{bmatrix} x \\ y \end{bmatrix} = A \begin{bmatrix} \hat{x}-m_x \\ \hat{y}-m_y \end{bmatrix} + \begin{bmatrix} m_x \\ m_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \tag{A.1}$$

where $[m_x\ m_y]^T$ is the centre of the transformation[2]. Solving equation (A.1) with respect to $(\hat{x},\hat{y})$ we obtain

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = A^{-1} \begin{bmatrix} x-m_x-t_x \\ y-m_y-t_y \end{bmatrix} + \begin{bmatrix} m_x \\ m_y \end{bmatrix}. \tag{A.2}$$

To determine the pixel value $I(x,y)$, we need to interpolate the pixels values in the source image $\hat{I}$ that are in the neighbourhood of $(\hat{x},\hat{y})$. The simplest interpolation method is the nearest neighbour interpolation, where we set $I(x,y)=\hat{I}(u,v)$ with (u,v) being the nearest pixel location from $(\hat{x},\hat{y})$. A more satisfactory interpolation is the bilinear interpolation which computes $I(x,y)$ as a weighted sum (with the weights being positive and summing to one) of the four nearest pixels of $(\hat{x},\hat{y})$.

The above procedure allows us to construct the underlying transformation matrix T that can equivalently perform the transformation through a matrix multiplication. Particularly, for each pixel location (x,y) there is a row T^p in the transformation matrix T computed as follows. When we use nearest neighbour interpolation, the row T^p has 1 in the column corresponding to the nearest neighbour of $(\hat{x},\hat{y})$ and 0 everywhere else. When we use bilinear interpolation, the row has at most four non-zero elements at appropriate locations that store the interpolation weights of the four neighbours of

[2]Note that if $[m_x\ m_y]^T$ is the origin of the pixels coordinates system (i.e. $[m_x\ m_y]^T=[0\ 0]^T$), then $[m_x\ m_y]^T$ disappears from all the above derivations.

$(\hat{x}, \hat{y})$. Clearly, we can store the transformation matrix T as a sparse matrix and thus using sparse algebra any matrix multiplication can be carried out fast in $O(P)$ time, where P is the number of rows of T.

The above procedure for computing transformation matrices is general and it can be applied to more complex transformations than affines such as projective transformations. We should point out that how much sparse the matrix T will be depends on the interpolation method used in the warping procedure and not on the type of transformation. Note that in all algorithms we have implemented for learning objects, we use nearest neighbour interpolation which is fast.

One can ask if we really need to worry about constructing a transformation matrix in the first place since we can compute the transformations in any way we wish e.g. as described above. It turns out that during the EM algorithm we need at least to perform a matrix multiplication of the transpose matrix T^T with some vector image $\mathbf{x}$, and generally computing $T^T\mathbf{x}$ without having stored the matrix T is not obvious. However, for simple translations in integer number of pixels, where T simply shifts the image forwards, we know that the transpose T^T shifts the image backwards, and thus in this case we can do all the computations without needed to express the matrix T.

As part of this appendix we have implemented a MATLAB function called *warping.m* that takes as input an image $\hat{I}$ and the affine transformation parameters $(A, [t_x\, t_y]^T)$ and returns the transformed image I and the corresponding matrix T. Also the user can choose between nearest neighbour or bilinear interpolation. This function is available from `http://www.dai.ed.ac.uk/homes/s0129556/PhDthesis`.

A.2 EM for learning one object against a static background

In this section we present the EM algorithm for the case we have one foreground object against a static background (see section 2.2.2). We also describe in detail all the differentiations that involve the transformation matrices and are required in the M-step of the EM algorithm. Note that the EM algorithm for learning one foreground object against clutter as well as the k-means described in section 2.2.1 will be expressed

similarly to the above EM algorithm.

We introduce first some notation. If $\mathbf{y}$ and $\mathbf{z}$ are two vectors of the same size, then $\mathbf{y} * \mathbf{z}$ defines the element-wise product between these vectors and $\mathbf{y} * \mathbf{y}$ is written as $\mathbf{y}^2$ for compactness. Similarly, the element-wise division between two vectors is denoted by $\mathbf{y}./\mathbf{z}$. A vector containing ones is denoted by $\mathbf{1}$. Also summations of the form $\sum_{p=1}^{P} y_p z_p$ are written in vector notation $\mathbf{y}^T\mathbf{z}$, e.g. $\mathbf{y}^T\mathbf{1}$ denotes the sum of elements of $\mathbf{y}$.

In our implementation the transformation matrices T_{j_f} of the foreground object have at most one 1 (and the other entries 0) in each row and correspond to translations or full affine transformations. Our derivations below regarding these matrices only require two constraints: (1) that the value of each element of $T_{j_f}\boldsymbol{\pi}$ is a valid probability (i.e. lies in $[0,1]$) and (2) that $\log(T_{j_f}\boldsymbol{\pi}) = T_{j_f}\log\boldsymbol{\pi}$ and $\log(\mathbf{1} - T_{j_f}\boldsymbol{\pi}) = T_{j_f}\log(\mathbf{1} - \boldsymbol{\pi})$, where $\log\mathbf{v}$ denotes the element-wise logarithm of a vector $\mathbf{v}$. These constraints certainly hold for matrices which have only one non-zero element in each row.

Assume we have a set of images $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ showing one movable object against a static background. To maximize the likelihood of the model discussed in 2.2.2, we need the EM algorithm to deal with the missing transformations j_f and the binary variables $\mathbf{s}$. The EM operates in the following expected complete data log likelihood (also call it Q function)

$$\begin{aligned} Q = \sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)\Big\{(\overline{\mathbf{s}}_{j_f}^n)^T(\log T_{j_f}\boldsymbol{\pi} + (-\frac{1}{2\sigma_f^2}(\mathbf{x}^n - T_{j_f}\mathbf{f})^2 - \frac{1}{2}\log\sigma_f^2\mathbf{1}) \\ + (\mathbf{1} - \mathbf{s}_{j_f}^n)^T(\log(\mathbf{1} - T_{j_f}\boldsymbol{\pi}) + (-\frac{1}{2\sigma_b^2}(\mathbf{x}^n - \mathbf{b})^2 - \frac{1}{2\sigma_b^2}\log\sigma_b^2\mathbf{1})\Big\} + const, \quad \text{(A.3)} \end{aligned}$$

where *const* denotes a constant term. In the E-step the posterior probability $P(j_f|\mathbf{x}^n)$ over the transformations and the vector $\overline{\mathbf{s}}_{j_f}^n$ are computed according to (2.10) and (2.11), respectively. In the M-step Q is maximized with respect to the parameters $\{\boldsymbol{\pi}, \mathbf{f}, \mathbf{b}, \sigma_f^2, \sigma_b^2\}$. It is straightforward to find the updates for the parameters $\{\mathbf{b}, \sigma_f^2, \sigma_b^2\}$ which are given by

$$\mathbf{b} \leftarrow \sum_{n=1}^{N}\left(\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)(\mathbf{1} - \overline{\mathbf{s}}_{j_f}^n)\right) * \mathbf{x}^n./\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)(\mathbf{1} - \overline{\mathbf{s}}_{j_f}^n), \qquad \text{(A.4)}$$

$$\sigma_f^2 \leftarrow \frac{\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[(\overline{\mathbf{s}}_{j_f}^n)^T(\mathbf{x}^n - T_{j_f}\mathbf{f})^2]}{\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[(\overline{\mathbf{s}}_{j_f}^n)^T\mathbf{1}]}, \tag{A.5}$$

$$\sigma_b^2 \leftarrow \frac{\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[(\mathbf{1} - \overline{\mathbf{s}}_{j_f}^n)^T(\mathbf{x}^n - \mathbf{b})^2]}{\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[(\mathbf{1} - \overline{\mathbf{s}}_{j_f}^n)^T\mathbf{1}]}. \tag{A.6}$$

Taking the derivatives of Q with respect to $\mathbf{f}$ and $\boldsymbol{\pi}$ is more complicated. To express the derivative with respect to $\mathbf{f}$ we write Q as

$$Q = -\frac{1}{2\sigma_f^2}\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)\left\{\sum_{p=1}^{P}(\overline{\mathbf{s}}_{j_f}^n)_p(x_p^n - (T_{j_f}^p)^T\mathbf{f})^2\right\} + const, \tag{A.7}$$

where $T_{j_f}^p$ is the pth row of the matrix T_{j_f} arranged as a column vector and *const* denotes all the terms not involving $\mathbf{f}$. If we differentiate the term $q_{j_f}^n = \sum_{p=1}^{P}(\overline{\mathbf{s}}_{j_f}^n)_p(x_p^n - T_{j_f}^p\mathbf{f})^2$ with respect to $\mathbf{f}$, then to express the whole derivative we only need to sum over n and j_f (weighted by $P(j_f|\mathbf{x}^n)$). Particularly, we have

$$\begin{aligned}\nabla_{\mathbf{f}}\, q_{j_f}^n &= 2\sum_{p=1}^{P}(\overline{\mathbf{s}}_{j_f}^n)_p T_{j_f}^p(x_p^n - (T_{j_f}^p)^T\mathbf{f}) = 2\sum_{p=1}^{P}(\overline{\mathbf{s}}_{j_f}^n)_p x_p^n T_{j_f}^p - 2\left(\sum_{p=1}^{P}(\overline{\mathbf{s}}_{j_f}^n)_p T_{j_f}^p(T_{j_f}^p)^T\right)\mathbf{f} \\ &= 2T_{j_f}^T(\overline{\mathbf{s}}_{j_f}^n * \mathbf{x}^n) - 2T_{j_f}^T S_{j_f}^n T_{j_f}\mathbf{f},\end{aligned} \tag{A.8}$$

where $S_{j_f}^n$ is a diagonal matrix having as diagonal elements the entries of the vector $\overline{\mathbf{s}}_{j_f}^n$. Now by taking the derivative of Q with respect to $\mathbf{f}$, using equation (A.8) and setting to zero we obtain

$$\left(\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T S_{j_f}^n T_{j_f}]\right)\mathbf{f} = \sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)T_{j_f}^T(\overline{\mathbf{s}}_{j_f}^n * \mathbf{x}^n),$$

or

$$\mathbf{f} = \left(\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T S_{j_f}^n T_{j_f}]\right)^{-1}\sum_{n=1}^{N}\sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)T_{j_f}^T(\overline{\mathbf{s}}_{j_f}^n * \mathbf{x}^n). \tag{A.9}$$

Note that equation (A.9) is the general solution for $\mathbf{f}$ and holds for any choice of the transformation matrices T_j. We can solve the above linear system fast without requiring to carry out Gauss elimination. Particularly, since each T_{j_f} matrix has at most a single 1 in each row the matrix $T_{j_f}^T S_{j_f}^n T_{j_f}$ becomes diagonal where the diagonal elements

grouped into a vector are given by the vector $T_{j_f}^T \overline{\mathbf{s}}_{j_f}^n$. Thus, equation (A.9) can be computed fast by taking the element-wise division of two vectors:

$$\mathbf{f} = \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T(\overline{\mathbf{s}}_{j_f}^n * \mathbf{x}^n)]./ \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T \overline{\mathbf{s}}_{j_f}^n]. \tag{A.10}$$

To derive the update for the mask $\boldsymbol{\pi}$ we write equation (A.3) as follows:

$$\begin{aligned} Q &= (\log \boldsymbol{\pi})^T \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n) T_{j_f}^T \overline{\mathbf{s}}_{j_f}^n + (\log(\mathbf{1}-\boldsymbol{\pi}))^T \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n) T_{j_f}^T (\mathbf{1} - \overline{\mathbf{s}}_{j_f}^n) \\ &+ const, \end{aligned} \tag{A.11}$$

where we used the fact that $\log(T_j \boldsymbol{\pi}) = T_{j_f} \log(\boldsymbol{\pi})$ and $\log(\mathbf{1} - T_{j_f}\boldsymbol{\pi}) = T_{j_f} \log(\mathbf{1} - \boldsymbol{\pi})$. By differentiating with respect to $\boldsymbol{\pi}$ and setting to zero we get

$$\boldsymbol{\pi} = \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T \overline{\mathbf{s}}_{j_f}^n]./ \sum_{n=1}^{N} \sum_{j_f=1}^{J_f} P(j_f|\mathbf{x}^n)[T_{j_f}^T \mathbf{1}]. \tag{A.12}$$

To express the EM algorithm for learning one foreground object against clutter (see section 2.2.1), the derivation are analogous to the above but simpler since now the background disappears. For example, the update for $\mathbf{f}$ is given as in equation (A.10) but with all $\overline{\mathbf{s}}_{j_f}^n$ be equal to $\mathbf{1}$. Particularly, the update for $\mathbf{f}$ is

$$\mathbf{f} = \sum_{n=1}^{N} \sum_{j=1}^{J} P(j|\mathbf{x}^n)[T_j^T \mathbf{x}^n]./ \sum_{n=1}^{N} \sum_{j=1}^{J} P(j|\mathbf{x}^n)[T_j^T \mathbf{1}]. \tag{A.13}$$

Note that in case the transformation matrices are exact permutations, $T_j^T \mathbf{1} = \mathbf{1}$, and thus the denominator in (A.13) will be equal to N, which explains equation (2.5). Deriving equation (2.6) is straightforward. Finally, for the k-means algorithm the update (2.1) that minimizes the error function (2.2) is derived similarly[3].

[3] It can be also shown that the k-means is derived by the EM algorithm by letting the variance of the spherical covariance matrix $\sigma_f^2 I$ go to 0. See Bishop (1995) at page 190 for a similar explanation.

Appendix B

Details of the greedy algorithm

As also mentioned in appendix A.2, in our implementation the transformation matrices of the foreground objects T_{j_ℓ} have at most one 1 (and the other entries 0) in each row and correspond to translations or full affine transformations.

B.1 Learning the background

Here we derive the EM algorithm for learning a static or moving background. Learning the background consists of the first stage of the greedy algorithm and is carried out by maximizing the following log likelihood:

$$L_b = \sum_{n=1}^{N} \log \sum_{j_b=1}^{J_b} P_{j_b} \prod_{p=1}^{P} \{\alpha_b N(x_p^n; (T_{j_b}\mathbf{b})_p, \sigma_b^2) + (1-\alpha_b) U(x_p^n)\}. \quad \text{(B.1)}$$

Clearly, this log likelihood corresponds to a mixture model (with J_b components) where the component densities are factorized over the pixels and each pixel density is a two-component mixture. Application of the EM is straightforward and we can easily show that the expected complete data log likelihood in the EM framework is:

$$Q_b = \sum_{n=1}^{N} \sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n) \left\{ (\mathbf{r}_{j_b}^n)^T \left(-\frac{1}{2\sigma_b^2} (\mathbf{x}^n - T_{j_b}\mathbf{b})^2 - \frac{1}{2} \log \sigma_b^2 \mathbf{1} \right) \right\} + const, \quad \text{(B.2)}$$

where $P(j_b|\mathbf{x}^n) = \frac{P_{j_b} p(\mathbf{x}^n|j_b)}{\sum_{i=1}^{J_b} P_i p(\mathbf{x}^n|i)}$ is the posterior probability of the transformation hidden variable j_b given the image $\mathbf{x}^n$ and $\mathbf{r}_{j_b}^n$ is a P length vector with the pth element storing

the probability according to which the pth pixel of image $\mathbf{x}^n$ is part of the non-occluded background given j_b:

$$(\mathbf{r}^n_{j_b})_p = \frac{\alpha_b N(x^n_p; (T_{j_b}\mathbf{b})_p, \sigma_b^2)}{\alpha_b N(x^n_p; (T_{j_b}\mathbf{b})_p, \sigma_b^2) + (1-\alpha_b)U(x^n_p)}. \tag{B.3}$$

In the E-step of the algorithm $P(j_b|\mathbf{x}^n)$ and $\mathbf{r}^n_{j_b}$ are obtained using the current parameter values. In the M-step the Q function is maximized with respect to the parameters $\{\mathbf{b}, \sigma_b^2\}$ giving the following update equations:

$$\mathbf{b} \leftarrow \sum_{n=1}^{N}\sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n)[T^T_{j_b}(\mathbf{r}^n_{j_b} * \mathbf{x}^n)]./\sum_{n=1}^{N}\sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n)[T^T_{j_b}\mathbf{r}^n_{j_b}], \tag{B.4}$$

$$\sigma_b^2 \leftarrow \frac{\sum_{n=1}^{N}\sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n)[(\mathbf{r}^n_{j_b})^T(\mathbf{x}^n - T_{j_b}\mathbf{b})^2]}{\sum_{n=1}^{N}\sum_{j_b=1}^{J_b} P(j_b|\mathbf{x}^n)[(\mathbf{r}^n_{j_b})^T\mathbf{1}]}. \tag{B.5}$$

The above equations provide an exact M-step. The update for the background appearance $\mathbf{b}$ is very intuitive. For example consider the case when $P(j_b|\mathbf{x}^n) = 1$ for $j_b = j^*$ and 0 otherwise. For pixels which are ascribed to non-occluded background (i.e. $(\mathbf{r}^n_{j_b})_p \simeq 1$) the values of $\mathbf{x}^n$ are transformed by $T^T_{j^*}$ which maps the $P_x \times P_y$ image $\mathbf{x}^n$ into a larger image of size $M_x \times M_y$ so that $\mathbf{x}^n$ is located in the position specified by j_b and the rest of image pixels are filled with zero values. Thus, the non-occluded pixels found in each training image are located properly into the big panorama image and averaged to produce $\mathbf{b}$.

Note also that in the special case where the background is static the effect of transformation j_b is removed from all update equations and the parameters $\mathbf{b}$ and σ_b^2 are updated according to $\mathbf{b} \leftarrow \sum_{n=1}^{N}(\mathbf{r}^n * \mathbf{x}^n)./\sum_{n=1}^{N}\mathbf{r}^n$ and $\sigma_b^2 \leftarrow \frac{\sum_{n=1}^{N}(\mathbf{r}^n)^T(\mathbf{x}^n-\mathbf{b})^2}{\sum_{n=1}^{N}(\mathbf{r}^n)^T\mathbf{1}}$, respectively.

For random backgrounds, the above EM algorithm is not needed. In this case we simply set $\mathbf{b}$ to the mean of all training images and σ_b^2 to the mean variance of all different pixel variances. These background parameters are kept fixed for later stages.

B.2 Learning the foreground objects

Assume that we have already found the background as described previously. At each next stage the greedy algorithm searches for a foreground object. Below we describe

how the ℓth foreground object is found, where $\ell = 1,\ldots,L$.

When we search for the ℓth object, the background as well as the $\ell-1$ foreground objects have been found in previous stages[1]. As explained in section 3.3.3 we learn the ℓth object by maximizing the objective function

$$F_\ell = \sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)\Big\{\sum_{p=1}^{P}(\mathbf{z}_{\ell-1}^n)_p \log[(T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p)+ (\mathbf{1}-T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_b(x_p^n;(T_{j_b^n}\mathbf{b})_p)] - \log Q^n(j_\ell)\Big\}. \quad \text{(B.6)}$$

The above maximization can be done by a variational EM algorithm. In the E-step we maximize F_ℓ with respect to the $Q^n(j_\ell)$ which gives

$$Q^n(j_\ell) \propto \exp\left\{\sum_{p=1}^{P}(\mathbf{z}_{\ell-1}^n)_p \log[(T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p) + (\mathbf{1}-T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_b(x_p^n;(T_{j_b^n}\mathbf{b})_p)]\right\}, \quad \text{(B.7)}$$

with $Q^n(j_\ell)$ normalized to sum to one. In the M-step we maximize F_ℓ with respect to the object parameters $\{\mathbf{f}_\ell,\boldsymbol{\pi}_\ell,\sigma_\ell^2\}$. For this maximization we need again the EM algorithm. The EM algorithm operates in the following Q function

$$Q_\ell = \sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)(\mathbf{z}_{\ell-1}^n)^T[\bar{\mathbf{s}}_{j_\ell}^n * \log T_{j_\ell}\boldsymbol{\pi}_\ell + (\mathbf{1}-\bar{\mathbf{s}}_{j_\ell}^n) * \log(\mathbf{1}-T_{j_\ell}\boldsymbol{\pi}_\ell) + \bar{\mathbf{s}}_{j_\ell}^n * \mathbf{r}_{j_\ell}^n * (-\frac{1}{2\sigma_\ell^2}(\mathbf{x}^n - T_{j_\ell}\mathbf{f}_\ell)^2 - \frac{1}{2}\log\sigma_\ell^2\mathbf{1})] + const, \quad \text{(B.8)}$$

where each element of the vector $\bar{\mathbf{s}}_{j_\ell}^n$ stores the value $\frac{(T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p)}{(T_{j_\ell}\boldsymbol{\pi}_\ell)_p p_{f_\ell}(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p)+(\mathbf{1}-(T_{j_\ell}\boldsymbol{\pi}_\ell)_p)p_b(x_p^n;(T_{j_b^n}\mathbf{b})_p)}$ expressing the probability the pixel to be part of the object. Each element of $\mathbf{r}_{j_\ell}^n$ stores the probability the pixel to be non-occluded: $(\mathbf{r}_{j_\ell}^n)_p = \frac{\alpha_f N(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p,\sigma_\ell^2)}{\alpha_f N(x_p^n;(T_{j_\ell}\mathbf{f}_\ell)_p,\sigma_\ell^2)+(1-\alpha_f)U(x_p^n)}$. The algorithm in the E-step computes the quantities, $Q^n(j_\ell)$, $\bar{\mathbf{s}}_{j_\ell}^n$ and $\mathbf{r}_{j_\ell}^n$ as described above and in the M-step we update the parameters as follows:

$$\boldsymbol{\pi}_\ell \leftarrow \sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)T_{j_\ell}^T[\mathbf{z}_{\ell-1}^n * \bar{\mathbf{s}}_{j_\ell}^n]./\sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)T_{j_\ell}^T\mathbf{z}_{\ell-1}^n, \quad \text{(B.9)}$$

[1] Of course when we search for the first object there will be no previously learned foreground objects.

$$\mathbf{f}_\ell \leftarrow \sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell) T_{j_\ell}^T[\mathbf{z}_{\ell-1}^n * \overline{\mathbf{s}}_{j_\ell}^n * \mathbf{r}_{j_\ell}^n * \mathbf{x}^n] ./ \sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell) T_{j_\ell}^T[\mathbf{z}_{\ell-1}^n * \overline{\mathbf{s}}_{j_\ell}^n * \mathbf{r}_{j_\ell}^n], \quad \text{(B.10)}$$

$$\sigma_\ell^2 \leftarrow \frac{\sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)(\mathbf{z}_{\ell-1}^n)^T[\overline{\mathbf{s}}_{j_\ell}^n * \mathbf{r}_{j_\ell}^n * (\mathbf{x}^n - T_{j_\ell}\mathbf{f}_\ell)^2]}{\sum_{n=1}^{N}\sum_{j_\ell=1}^{J_f} Q^n(j_\ell)(\mathbf{z}_{\ell-1}^n)^T[\overline{\mathbf{s}}_{j_\ell}^n * \mathbf{r}_{j_\ell}^n]}. \quad \text{(B.11)}$$

As with the updates for $\mathbf{b}$ and σ_b^2 these updates make intuitive sense. Consider, for example, the ℓth appearance model $\mathbf{f}_\ell$ when $Q^n(j_\ell) = 1$ for $j_\ell = j^*$ and 0 otherwise. For pixels which are ascribed to the ℓth foreground and are not occluded (i.e. $(\mathbf{z}_{\ell-1}^n * \overline{\mathbf{s}}_{j^*}^n * \mathbf{r}_{j^*}^n)_p \simeq 1$), the values in $\mathbf{x}^n$ are transformed by $T_{j^*}^T$ (which is $T_{j^*}^{-1}$ in case the transformations are permutation matrices). This removes the effect of the transformation and thus allows the foreground pixels found in each training image to be averaged to produce $\mathbf{f}_\ell$.

B.3 Computation of the occlusion ordering

As explained in section 3.4, a naive computation of the occlusion ordering for L foreground objects in an image takes $L!$ time. This can be very computationally expensive, e.g. for $L = 7$ there are 5040 permutations of the foreground layers that we must consider. Here we discuss how we can speed up this computation and thus deal with large numbers of objects.

The key idea to efficiently compute the occlusion ordering of the objects is to identify how the objects overlap with each other in the observed image. It will be intuitive to explain this point with an example. Consider the images shown in the top row of Figure B.1, which have been also used in the Experiment 4 in section 3.6. These images show five objects and particularly a *floppy*, a *cd*, a *passport*, a *marker* and a *watch*. Now in Figure B.1b the *floppy* overlaps with the *cd* and the *marker* but does not overlap with the *passport* and the *watch*. Similarly, the *watch* and the *passport* overlap only with each other. By exploiting this property we can factorize the computation of the overall occlusion ordering of the objects. Roughly speaking, it is possible only by taking permutations between groups of overlapping objects to infer the overall occlusion ordering in the image. Next, we describe how we achieve this.

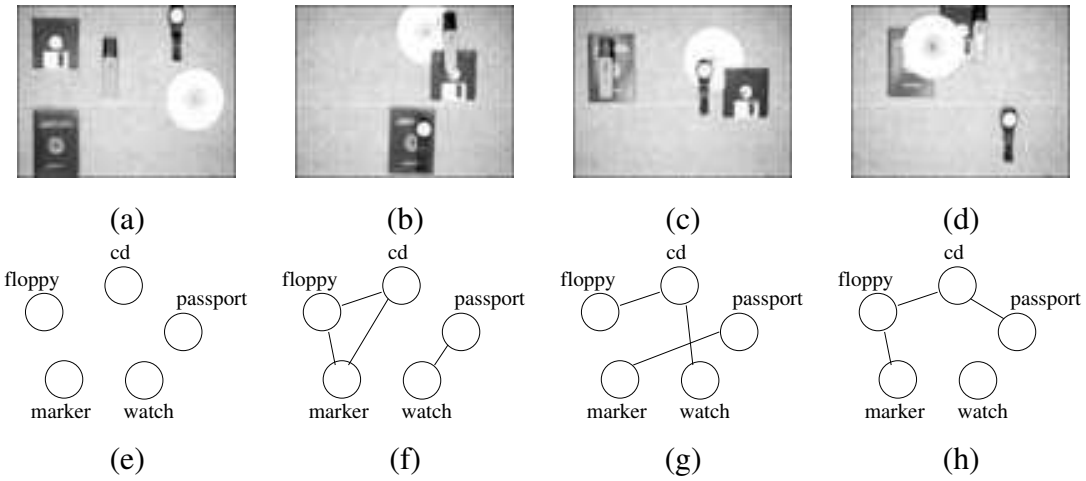


Figure B.1: The panels in the top row, i.e. (a) to (d), show 4 images of five objects (a $floppy$, a cd, a $passport$, a $marker$ and a $watch$) that can arbitrary move and occlude one another. The panels in the bottom row show the corresponding overlap graphs that describe how the objects interact with each other (either occlude or are occluded). Particularly, the graphs in each column correspond to each other, e.g. panel (e) shows the overlap graph for the image (a) etc.

First of all, for each object ℓ we identify the set of overlapping objects denoted by $\mathcal{N}(\ell)$. This can be easily done by first making each mask $\boldsymbol{\pi}_\ell$ binary, expressing the transformed masks (using the known transformations that instantiate the objects in the image) and checking which objects overlap with each other[2]. For example, for the Figure B.1b the overlap set will be $\mathcal{N}(floppy) = \{cd, marker\}$, $\mathcal{N}(cd) = \{floppy, marker\}$, $\mathcal{N}(watch) = \{passport\}$, etc. Equivalently these overlap sets define an undirected graph, where the nodes are the objects and edges indicate overlapping objects. For example, the graph that corresponds to Figure B.1b is shown in Figure B.1f.

Now, it can be shown that the occlusion ordering with the maximum log likelihood (and assuming binary masks) can be found by taking relative permutations of the objects within the same clique (a maximal complete subgraph). We omit a formal

[2]Particularly, we first compute all the transformed masks: $(T_{j_1}\boldsymbol{\pi}_1, \ldots, T_{j_L}\boldsymbol{\pi}_L)$. Then, we determine if any pair (k,l) of objects overlap by checking if the pixel-wise sum $T_{j_1}\boldsymbol{\pi}_k + T_{j_l}\boldsymbol{\pi}_l$ has at least one element with the value 2.

derivation of this result and simply state the steps of the algorithm: i) find the overlap set $\mathcal{N}(\ell)$, for any object ℓ using the transformed binary masks, ii) find the cliques in the underlying graph, iii) for any clique in graph take all the permutations of the objects within the clique and choose the one with the maximum log likelihood measured by ignoring all the image pixels that belong to the background and to objects out of the clique and iv) find the overall occlusion ordering by fusing the within-cliques orderings using the connectivity of the different cliques.

The complexity of finding the occlusion ordering is $O(G!)$ where G is the maximum clique size over all cliques in the graph. Let us see some examples of how the algorithm works. For the image of Figure B.1a, the corresponding graph is shown in Figure B.1e. Clearly all cliques have size one, thus we don't have to make any computation and all the occlusion orderings of the objects are equally suitable. For the image of Figure B.1b, the cliques in the corresponding graph shown in Figure B.1f are $C_1 = \{floppy, cd, marker\}$ and $C_2 = \{passport, watch\}$. The relative occlusion ordering of the first clique is $O_1 = \{marker, floppy, cd\}$ and for the second clique is $O_2 = \{watch, passport\}$. Since the two cliques are not connected we can fuse the relative orderings O_1 and O_2 in several equivalent ways, e.g. two (out of many) possible orderings are $O = \{marker, floppy, cd, watch, passport\}$ and $O = \{marker, floppy, watch, cd, passport\}$. Note that the number of permutations needed for this examples is $3! + 2! = 8$, while the naive computation needs $5! = 120$ permutations. Similarly we can work for the graphs in Figure B.1g and B.1h.

Appendix C

Focused search as variational EM

In section 4.2 we mentioned that once the tracking algorithm provides an approximation to the sequence of transformations that explain the motion of an object, the object model is learned using a focused search around these transformations. Here we describe this focused search algorithm.

We will consider only the case we learn the background, since the case of the foreground objects is treated similarly. To learn the background the greedy algorithm maximizes the log likelihood (B.1) with an exact EM as described in appendix B.1. Now we need to modify this algorithm so that to take into account the fact that tracking provides with some approximate transformations $(j_b^1, \ldots, j_b^N)$ for all training images. We express a lower bound (using Jensen's inequality) of the log likelihood (B.1):

$$\begin{aligned} F_b &= \sum_{n=1}^{N} \sum_{j_b=1}^{J_b} Q^n(j_b) \Bigg\{ \log \Bigg\{ P_{j_b} \prod_{p=1}^{P} \{\alpha_b N(x_p^n; (T_{j_b}\mathbf{b})_p, \sigma_b^2) + (1-\alpha_b) U(x_p^n)\} \Bigg\} \\ &- \log Q^n(j_b) \Bigg\}, \end{aligned} \tag{C.1}$$

which is true for any choice of the distributions $Q^n(j_b)$. Since tracking gives us an approximation of the transformation j_b^n of the background in the image $\mathbf{x}^n$, we assume that the true unknown transformation exists in the neighbourhood of j_b^n. We express this by considering a set of neighbouring transformation $\mathcal{N}(j_b^n)$ around j_b^n and con-

straining $Q^n(j_b) = 0$ for any $j_b \notin \mathcal{N}(j_b^n)$. Thus, the lower bound becomes

$$F_b = \sum_{n=1}^{N} \sum_{j_b \in \mathcal{N}(j_b^n)} Q^n(j_b) \left\{ \log \left\{ P_{j_b} \prod_{p=1}^{P} \{\alpha_b N(x_p^n; (T_{j_b}\mathbf{b})_p, \sigma_b^2) + (1-\alpha_b) U(x_p^n)\} \right\} - \log Q^n(j_b) \right\}, \tag{C.2}$$

which is the function that is maximized by EM (in the E-step over Q^ns and in the M-step over the parameters $\mathbf{b}$ and σ_b^2). Note that the case we do not consider any search at all is when each neighbouring set $\mathcal{N}(j_b^n)$ contains only j_b^n.

Bibliography

Attias, H. (2000). A variational Bayesian framework for graphical models. In *Advances in Neural Information Processing Systems 12*. MIT Press.

Barlow, H. (1989). Unsupervised Learning. *Neural Computation*, 1:295–311.

Bergen, J. R., Burt, P. J., Hingorani, R., and Peleg, S. (1992). A Three-Frame Algorithm for Estimating Two-Component Image Motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):886–896.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford.

Black, M. and Jepson, A. (1996). EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation. *Proc. ECCV*, pages 329–342.

Black, M. J. and Anandan, P. (1996). The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields. *Computer Vision and Image Understanding*, 63(1):75–104.

Bregler, C. and Omohundro, S. (1994). Surface learning with application to lipreading. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 6*.

Brunelli, R. and Poggio, T. (1995). Template matching: Matched spatial filters and beyond. Technical report, CBCL Paper 123/AI Memo 1549, Massachusetts Institute of Technology, Cambridge.

Burl, M. C., Weber, M., and Perona, P. (1998). A probabilistic approach to object recognition using local photometry and global geometry. In *Proceedings in ECCV*, pages 628–641.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D. (1993). AutoClass: A Bayesian Classification System. In Buchanan, B. G. and Wilkins, D. C., editors, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, pages 431–441. Kaufmann, San Mateo, CA.

Cootes, T. F., Taylor, C., Cooper, D., and Graham, J. (1992). Training models of shape from sets of examples. In *Proc. BMVC*, pages 9–18. Springer-Verlag.

Csurka, G., Bray, C., Dance, C., and Fan, L. (2004). Visual categorization with bags of keypoints. In *ECCV workshop on Statistical Learning in Computer Vision*, pages 59–74.

Darrell, T. and Pentland, A. P. (1995). Cooperative Robust Estimation Using Layers of Support. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):474–487.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39(1):1–38.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2000). Efficient Matching of Pictorial Structures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2000*, pages II:66–73.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79.

Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 264–271.

Figueiredo, M. A. T. and Jain, A. K. (2002). Unsupervised learning of finite mixture models. *PAMI*, 24(3):381–396.

Fischler, M. A. and Elschlager, R. A. (1973). The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 21(1):67–92.

Fitzgibbon, A. and Zisserman, A. (2002). On Affine Invariant Clustering and Automatic Cast Listing in Movies. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Proceedings of the Seventh European Conference on Computer Vision, ECCV 2002*, pages III 304–320. Springer. Lecture Notes in Computer Science 2353.

Forsyth, D., Malik, J., Fleck, M., and Ponce, J. (1997). Primitives, Perceptual Organization and Object Recognition. Technical report, Beckman Institute, University of Illinois.

Forsyth, D. A. and Ponce, J. (2003). *Computer Vision: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey.

Frey, B., Hinton, G., and Dayan, P. (1996). Does the wake-sleep algorithm produce good density estimators. In Touretsky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems 8*. MIT Press.

Frey, B. J. and Jojic, N. (1999). Estimating mixture models of images and inferring spatial transformations using the EM algorithm. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 1999*. IEEE Computer Society Press. Ft. Collins, CO.

Frey, B. J. and Jojic, N. (2001). Fast, Large-Scale Transformation-Invariant Clustering. In *Advances in Neural Information Processing Systems 14*. MIT press.

Frey, B. J. and Jojic, N. (2003). Transformation Invariant Clustering Using the EM Algorithm. *IEEE Trans Pattern Analysis and Machine Intelligence*, 25(1):1–17.

Frey, B. J. and Jojic, N. (2004). Advances in algorithms for inference and learning in complex probability models. *To appear in IEEE Trans Pattern Analysis and Machine Intelligence*.

Ghahramani, Z. (1995). Factorial Learning and the EM Algorithm. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 617–624. Morgan Kaufmann, San Mateo, CA.

Ghahramani, Z. and Jordan, M. (1997). Factorial hidden markov models. *Machine Learning*, 29:245–275.

Green, P. J. (1995). Reversible Jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732.

Grenander, U., Chow, Y., and Keenan, D. M. (1991). Hands: A pattern theoretic study of biological shapes.

Heap, A. J. and Hogg, D. C. (1995). Extending the point distribution method using polar coordinates. *Image and Vision Computing*, 14(8):589–599.

Heisele, B., Serre, T., Pontil, M., Vetter, T., and Poggio, T. (2002). Categorization by learning and combining object parts. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann.

Horn, B. K. P. and Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203.

Ioffe, S. and Forsyth, D. A. (2001). Probabilistic methods for finding people. In *International Journal of Computer Vision*, pages 45–68.

Irani, M., Rousso, B., and Peleg, S. (1994). Computing Occluding and Transparent Motions. *International Journal of Computer Vision*, 12(1):5–16.

Jepson, A. and Black, M. (1993). Mixture models for optical flow computation. In *Partitioning Data Sets DIMACS Workshop*, pages 271–286.

Jepson, A. D., Fleet, D. J., and Black, M. J. (2002). A Layered Motion Representation with Occlusion and Compact Spatial Support. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Proceedings of the Seventh European Conference on Computer Vision, ECCV 2002*, pages I 692–706. Springer. Lecture Notes in Computer Science 2353.

Jojic, N., Frey, B., and Kannan, A. (2003). Learning appearance and transparency manifolds of occluded objects in layers. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*.

Jojic, N. and Frey, B. J. (2001). Learning Flexible Sprites in Video Layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2001*. IEEE Computer Society Press. Kauai, Hawaii.

Jojic, N., Petrovic, N., Frey, B. J., and Huang, T. S. (2000). Transformed hidden Markov models: Estimating mixture models of images and inferring spatial transformations in video sequences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kanizsa, G. (1979). *Organization in Vision: Essays on Gestalt Perception*. New York: Praeger.

Kannan, A., Jojic, N., and Frey, B. (2005). Generative model for layers of appearance and deformation. In *10th International Workshop on Artificial Intelligence and Statistics*.

Koenderink, J. J. and van Doorn, A. J. (1979). The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216.

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

McLachlan, B. G. and Peel, D. (2000). *Finite Mixture Models*. Wiley, New York.

Meek, C., Thiesson, B., and Heckerman, D. (2002). Staged Mixture Modelling and Boosting. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 335–343, San Francisco, CA. Morgan Kaufmann Publishers.

Meila, M. and Heckerman, D. (2001). An experimental comparison of model-based clustering methods. *Machine Learning*, 42:9–29.

Murase, H. and Nayar, S. (1995). Visual learning and recognition of 3D objects from appearance. *International Journal of Computer Vision*, 14(1):5–24.

Neal, R. and Hinton, G. (1998). A view of the EM algorithm that justifies incremental, sparse and other variants. In Jordan, M., editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., and Poggio, T. (1997). Pedestrian detection using wavelet templates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 193–199.

Poggio, T. and Beymer, D. (1996). Regularization networks for visual learning. In *Early Visual Learning*, pages 43–67.

Rissanen, J. (1987). Stochastic Complexity and the MDL Principle. *Econometric Reviews*, 6:85–102.

Ross, D. A. and Zemel, R. S. (2003). Multiple Cause Vector Quantization. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

Rosset, S. and Segal, E. (2003). Boosting density estimation. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

Rowe, S. and Blake, A. (1995). Statistical Background Modelling For Tracking With A Virtual Camera. In Pycock, D., editor, *Proceedings of the 6th British Machine Vision Conference*, volume volume 2, pages 423–432. BMVA Press.

Roweis, S. (1998). EM algorithms for PCA and SPCA. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural Network-based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38.

Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473.

Saund, E. (1995). A multiple cause mixture model for unsupervised learning. *Neural Computation*, 7:51–71.

Sawhney, H. S. and Ayer, S. (1996). Compact Representations of Videos Through Dominant and Multiple Motion Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):814–830.

Schneiderman, H. and Kanade, T. (2004). Object Detection Using the Statistics of Parts. *International Journal of Computer Vision*, 56(3):151–177.

Schwarz, G. (1978). Estimating the Dimension of a Model. *Annals of Statistics*, 6:461–464.

Shams, L. and von der Malsburg, C. (1999). Are object shape primitives learnable? *Neurocomputing*, 26-27:855–863.

Sirovich, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of Optical Society of America*, 4(3):519–524.

Tao, H., Sawhney, H. S., and Kumar, R. (2000). Dynamic Layer Representation with Applications to Tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages II:134–141.

Thollard, F., Sebban, M., and Ezequel, P. (2002). Boosting density function estimators. In *13th European Conference on Machine Learning*, pages 431–443.

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11:443–482.

Titsias, M. K. and Williams, C. K. I. (2004). Fast unsupervised greedy learning of multiple objects and parts from video. In *Proc. Generative-Model Based Vision Workshop*.

Torr, P. H. S. (1998). Geometric motion segmentation and model selection. *Phil. Trans. Roy. Soc. Lond. A*, 356:1321–1340.

Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86.

Ullman, S. (1996). *High-Level Vision: Object Recognition and Visual Cognition*. MIT Press.

Ullman, S., Vidal-Naquet, M., and Sali, E. (2002). Visual features of intermediate complexity and their use in classification. *Nature Neuroscience*, 5(7):1–6.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley and Sons.

Vasconcelos, N. and Lippman, A. (2001). Empirical Bayesian Motion Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):217–221.

Verbeek, J., Vlassis, N., and Krose, B. (2003). Efficient greedy learning of Gaussian mixture models. *Neural Computation*, 15:469–485.

Vlassis, N. and Likas, A. (2002). A greedy EM for Gaussian mixture learning. *Neural Processing Letters*, 15:77–87.

Wang, J. Y. A. and Adelson, E. H. (1994). Representing Moving Images with Layers. *IEEE Transactions on Image Processing*, 3(5):625–638.

Weber, M., Welling, M., and Perona, P. (2000). Unsupervised Learning of Models for Recognition. In *Proceedings of the Fifth European Conference on Computer Vision, ECCV 2000*, pages 18–32.

Weiss, Y. and Adelson, E. (1996). A unified mixture framework for motion segmentation: incorporating spatial coherence and estimating the number of models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Wertheimer, M. (1923). Laws of organization in perceptual forms. *Pscycol. Forsch.*, 4:301–350. English translation in: W.B. Ellis, A source book of Gestalt (1973) 71-88.

Williams, C. K. I. (1994). Combining deformable models and neural networks for handprinted digit recognition. PhD thesis, Department of Computer Science, University of Toronto.

Williams, C. K. I. and Titsias, M. K. (2004). Greedy Learning of Multiple Objects in Images using Robust Statistics and Factorial Learning. *Neural Computation*, 16(5):1039–1062.

Williams, C. K. I. and Titsias, M. T. (2003). Learning about multiple objects in images: Factorial learning without factorial search. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems*. MIT Press.

Wills, J., Agarwal, S., and Belongie, S. (2003). What Went Where. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2003*, pages I:37–44.

Winn, J. and Blake, A. (2005). Generative Affine Localisation and Tracking. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*. MIT Press.